



**Guia de programação
Javascript**

ISB Indústria e Comércio

Versão

10/05/2011

Índice

Introdução.....	3
A linguagem Javascript.....	3
Java e Javascript são a mesma coisa?.....	3
Comandos.....	3
Criando Variáveis.....	3
Variáveis globais.....	3
Variáveis locais.....	3
Inserir comentários.....	4
Strings.....	4
Controles Especiais.....	4
Operadores.....	4
Operadores Aritméticos.....	4
Operadores de atribuição.....	5
Operadores Lógicos.....	5
Operadores de string.....	6
Outros operadores.....	6
Comandos Condicionais.....	6
Comando IF.....	6
Comando FOR.....	7
Comando WHILE.....	7
Operação Ternária.....	7
Criando Funções.....	8
Funções nativas.....	8
Objetos.....	8
Criando Novas Instâncias.....	9
Criando Novos Objetos.....	9
Objeto Math.....	9
Objeto Array.....	10
Objeto String.....	12
Javascript no AutoMAC.....	12
Objetos para controle de AutoMAC's.....	12
Objeto device.....	13
Objeto io(0-13).....	13
Objeto da(0-1).....	13
Objeto ad(0-7) e ad_int.....	14
Objeto timer.....	14
Objeto counter.....	14

Introdução

Neste manual iremos abordar o uso da linguagem Javascript que é utilizada no “*Terminal de Comandos*” da interface para controle de *AutoMAC*'s, sendo uma linguagem simples, muito conhecida e utilizada para processamento de dados.

A linguagem Javascript

Por ser uma linguagem muito difundida na internet, principalmente para inserção de lógicas em páginas Web, o Javascript passa a ser usado cada vez mais em diversas aplicações, por este motivo abordaremos o seu uso em eventos de um AutoMAC.

Passaremos a ver neste documento como controlar as mais variadas ações, inicialmente abordando as definições de variáveis, funções e declaração de objetos, e finalmente vendo que objetos servem de controle do AutoMAC. Tendo esse conhecimento em mãos, podemos tomar ações conforme a ativação de eventos, inserindo lógica na tomada de decisões pelo sistema.

Java e Javascript são a mesma coisa?

Não! Java e Javascript são duas linguagens completamente diferentes em conceito e design. Sendo Java uma linguagem mais complexa, se encontrando na mesma categoria de outras linguagens de programação como C e C++.

Comandos

Os comandos em Javascript diferem letras maiúsculas e minúsculas em sua sintaxe, portanto, é necessário que seja obedecida a forma de escrever os comandos, de acordo com a forma apresentada ao longo deste manual. Caso seja cometido algum erro de sintaxe quando da escrita de um comando, o Javascript interpretará, o que seria um comando, como sendo o nome de uma variável ou informará algum erro de sintaxe.

Criando Variáveis

As variáveis são criadas automaticamente, simplesmente pela associação de um valor a elas.

Ex.: NovaVariavel = 0;

Existem dois tipos de definições de variáveis, das quais podem ser globais ou locais, seu uso dependerá muito do tipo de implementação utilizada, abaixo iremos dar uma explicação mais detalhada sobre cada uma delas.

Variáveis globais

As variáveis globais podem ser acessadas por qualquer parte do código, quando precisamos referenciar variáveis por variadas partes do código, estas por tanto devem ser definidas como globais.

Variáveis locais

As variáveis locais geralmente são criadas dentro de uma função e para que elas sejam realmente locais devem ser procedidas pela palavra-chave var.

Ex.: var variavel = 0;

Embora não seja recomendável, em uma função, pode ser definida uma variável local com o mesmo nome de uma variável global.

Inserir comentários

Muitas vezes os códigos gerados deverão ser passados por várias pessoas, muitas das quais poderão não entender completamente a lógica utilizada, ou só iremos ver esse mesmo código depois de muito tempo, para isso devemos expor comentários juntamente ao código para sabermos o que ele faz. Para isso existem dois tipos de definição de comentários para códigos Javascript que são:

// Comentário de linha – é válido até que o fim da linha seja atingido

/*...*/ - Delimitadores para inserir um texto com mais de uma linha como comentário.

Strings

Strings nada mais são do que uma cadeia de caracteres, que representam textos a serem demonstrados ao usuário. Os delimitadores para uma string são " ou ', caso seja necessário a utilização destes caracteres como parte da string, utilize \ precedendo " ou '.

Controles Especiais

São representações especiais de caracteres, os mais comuns podem ser vistos abaixo.

\b – backspace

\n – new line

\r – carriage return

\t – tab characters

Ex.: print ("Cuidado com o uso de \" ou \' em uma string\n")

Operadores

São símbolos, em geral de um ou poucos caracteres, que permitem operações aritméticas, lógicas, etc. Em outras palavras, são usados basicamente para modificar valores de variáveis. Nesta página são dados os operadores de Javascript em forma de tabelas classificadas por grupo de funções.

Operadores Aritméticos

São operadores a serem utilizados em cálculos, referências de indexadores. Ao longo do manual estes operadores serão largamente utilizados, dando uma noção mais precisa do seu potencial.

Operador	Descrição	Exemplo
+	adição de valor	a = 2 + 3; b = b + 1;
-	subtração de valores ou como valor unário	x = a - b; x = -x; x = -(a + b);
*	multiplicação de valores	a = 2 * 3; b = c * 5;
/	divisão de valores	a = 50 / 3;
%	obtem o resto de uma divisão:	d = 5 % 3; d assume valor 2.
++var	Incremento de 1 (antes).	Se x é 2, y = ++x faz x igual a 3 e depois y igual a 3.

var++	Incremento de 1 (depois).	Se x é 2, y = x++ faz y igual a 2 e depois x igual a 3.
--var	Decremento de 1 (antes).	Se x é 2, y = --x faz x igual a 1 e depois y igual a 1.
var--	Decremento de 1 (depois).	Se x é 2, y = x-- faz y igual a 2 e depois x igual a 1.

Operadores de atribuição

São operadores aritméticos utilizados na atribuição de variáveis.

Operador	Descrição	Exemplo
=	Atribui o valor do operando esquerdo ao operando direito.	x = 3; a = b + c;
+=	Soma 2 valores e atribui o resultado ao primeiro valor.	x += 3; Se x era 1, passa para 4.
-=	Subtrai 2 valores e atribui o resultado ao primeiro.	x -= 3; Se x era 1, passa para -2.
*=	Multiplica 2 valores e atribui o resultado ao primeiro.	x *= 2; Se x era 4, passa para 8.
/=	Divide 2 valores e atribui o resultado ao primeiro.	x /= 2; Se x era 4, passa para 2.
%=	Calcula o resto da divisão de 2 valores e atribui o resultado ao primeiro.	x %= 2; Se x era 3, passa para 1.

Operadores Lógicos

São operadores a serem utilizados em comandos condicionais, tais como: IF , FOR e WHILE. Os comandos condicionais serão vistos mais a frente.

Operador	Descrição	Exemplo(s), supondo a = 3 e b = 5
==	Verdadeiro se os operandos são iguais. Se não são do mesmo tipo, a linguagem tenta converter para a correta comparação.	a == 3; // retorna verdadeiro a == b; // retorna falso
!=	Verdadeiro se os operandos não são iguais. Se não são do mesmo tipo, a linguagem tenta converter para a correta comparação.	a != 3; // retorna falso a != b; // retorna verdadeiro
===	Verdadeiro se os operandos são iguais e do mesmo tipo.	a === 3; // retorna verdadeiro a === "3"; // retorna falso
!==	Verdadeiro se os operandos não são iguais ou não são do mesmo tipo.	a !== b; // retorna verdadeiro a !== "3"; // retorna verdadeiro
>	Verdadeiro se o operando esquerdo é maior que o direito.	a > b; // retorna falso b > a; // retorna verdadeiro
>=	Verdadeiro se o operando esquerdo é maior ou igual ao direito.	a >= 3; // retorna verdadeiro b >= 7; // retorna falso
<	Verdadeiro se o operando esquerdo é menor que o direito.	a < b; // retorna verdadeiro b < a; // retorna falso
<=	Verdadeiro se o operando esquerdo é menor ou igual ao direito.	a <= 3; // retorna verdadeiro a <= 0; // retorna falso
&&	E lógico – retorna verdadeiro se os operandos forem verdadeiros.	a <= 3 && b > 5; // retorna falso a > 0 && a < 6; // retorna verdadeiro
	Ou lógico – retorna verdadeiro se um dos operandos for verdadeiro.	a > 3 b > 6; // retorna falso a <= 3 a <= 0; // retorna verdadeiro

!	NÃO lógico: retorna verdadeiro se o operando é falso e vice-versa	! (a==3) // retorna falso ! (a!=3) // retorna verdadeiro
----------	-------------------------------------------------------------------	-------------------------------------------------------------

Operadores de string

São operadores que servem para manipulação de strings.

Operador	Descrição	Exemplo(s)
+	Concatenar strings.	str_1 = "Bom"; str_2 = str_1 + " dia"; str_2 contém "Bom dia"
+=	Concatenar e atribuir o resultado ao operando da esquerda.	str_1 = "Bom"; str_1 += " dia"; str_1 contém "Bom dia"

Outros operadores

Existem outros tipos de operadores no Javascript que não influenciam diretamente valores de variáveis, somente fazem referencia a um objeto ou ação que pode ser feita no mesmo.

Operador	Descrição	Exemplo(s)
<u>delete</u>	Exclui um objeto, uma propriedade de objeto, um elemento de uma array ou uma variável explicitamente declarada com var. Para elemento de array, não altera o seu tamanho. Apenas o elemento fica indefinido.	val = new Array(10); delete val[4]; delete val; var nivel = 20; delete nivel;
<u>new</u>	Cria um novo objeto entre aqueles já existentes na linguagem (str = new String("abc"), etc) ou cria um objeto definido pelo usuário.	function prod(nome,unidade,valor){ this.nome = nome; this.unidade = unidade; this.valor = valor; } P01 = new prod("banana","kg",2.00);
<u>this</u>	Faz referência ao objeto em uso.	function Teste() { this.i = 10; } var teste = new Teste(); print(teste.i);
<u>typeof</u>	Retorna uma string indicativa do tipo de operando.	var val = 10; typeof val retorna "number" str = "abc"; typeof str retorna "string"
<u>void</u>	Indica uma expressão para ser avaliada mas sem retornar nenhum valor e também serve para gerar uma variável com valor undefined.	a = void (b = 5, c = 7); print('a = ' + a + ' b = ' + b + ' c = ' + c);

Comandos Condicionais

São comandos que condicionam a execução de uma certa tarefa à veracidade ou não de uma determinada condição, ou enquanto determinada condição for verdadeira. São eles:

Comando IF

Serve para avaliar uma expressão, se a expressão for verdadeira ela irá executar um bloco de ação, mas opcionalmente poderá executar outra operação caso seja falso.

```
if (condição)
    { ação para condição satisfeita }
```

```
[ else
    { ação para condição não satisfeita } ]
```

Ex.

```
if (Idade < 18)
    {Categoria = "Menor" }
else
    {Categoria = "Maior"}
```

Comando FOR

Esse comando é utilizado para repetir um bloco de ações e contar a quantidade de repetições conforme determinado. O comando aceita três tipos de parâmetros, de inicialização por onde as variáveis são inicializadas, condição para continuação onde o bloco é executado enquanto a condição for verdadeira, e o parâmetro para operações de incremento e controle.

```
for ( [inicialização/criação de variável de controle] ;
    [condição] ;
    [incremento da variável de controle] )
{
    ação
}
```

Ex.

```
for (x = 0 ; x <= 10 ; x++)
{ print("X igual a " + x) }
```

Comando WHILE

Executa uma ação enquanto determinada condição for verdadeira.

```
while (condição)
{ ação }
```

Ex.

```
var contador = 10
while (contador > 1)
{ contador-- }
```

Operação Ternária

É um operador condicional, que serve para avaliar uma condição e retornar um tipo de valor caso a condição seja verdadeira ou falsa.

```
receptor = ( (condição) ? verdadeiro : falso)
```

```
Ex.: NomeSexo = ((VarSexo == "M") ? "Masculino" : "Feminino")
```

OBS:

Nos comandos FOR e WHILE a diretiva "break" pode ser utilizada para interromper a condição principal e sair do loop. Da mesma forma, a diretiva "continue" interrompe uma ação (se determinada condição ocorrer) mas volta para o loop.

Diretivas/condições entre [] significam que são opcionais.

Criando Funções

Uma função é um set de instruções, que só devem ser executadas quando a função for acionada. A sintaxe geral é a seguinte:

```
function NomeFuncao (Parâmetros)
    { Ação }
```

Suponha uma função que tenha como objetivo informar se uma pessoa é maior ou menor de idade, recebendo como parâmetro a sua idade.

```
function Idade (Anos) {
    if (Anos > 17)
        { alert ("Maior de Idade") }
    else
        { alert ("menor de Idade") }
}
```

Para acionar esta função, suponha uma caixa de texto, em um formulário, na qual seja informada a idade e, a cada informação, a função seja acionada.

Funções nativas

São funções embutidas na própria linguagem. A sintaxe geral é a seguinte:

Result = função (informação a ser processada)

- **eval** = Calcula o conteúdo da string
- **parseInt** - Transforma string em inteiro
- **parseFloat** - Transforma string em número com ponto flutuante

Ex. 1: Result = eval("(10 * 20) + 2 - 8")

Ex. 2: Result = eval(string)

No primeiro exemplo Result seria igual a 194. No segundo, depende do conteúdo da string, que também pode ser o conteúdo (value) de uma caixa de texto.

Objetos

O Javascript define certos tipos de objetos para operações básicas da linguagem, facilitando as suas operações através de vários métodos pré definidos. Abaixo iremos ver os principais objetos definidos na linguagem, que poderão auxiliar em diversos tipos de operações, e também procederemos com a instanciação de novos objetos.

Criando Novas Instâncias

Através do operador *new* podemos criar novas instâncias de objetos já existentes, possibilitando a alteração do seu conteúdo, porém, mantendo as suas propriedades. A sintaxe geral deve ser a seguinte:

```
NovoObjeto = new ObjetoExistente(parâmetros);
```

```
Ex1.: MinhaString = new String()
```

MinhaString passou a ser um objeto tipo String, sem nenhum conteúdo definido.

```
Ex2.: MinhaString = new String("Fazendo um teste")
```

MinhaString passou a ser um objeto tipo String, porém, com o conteúdo iniciado com "Fazendo um teste".

Criando Novos Objetos

Podemos definir nossos próprios tipos de objetos, aos quais podem agrupar um conjunto de operações que desejamos executar, a declaração de um objeto nada mais é do que criarmos uma função ao qual deve ter suas propriedades e métodos precedidos pelo operador *this*.

Ex.: Suponha a existência do seguinte objeto chamado Empresas.

```
function Empresas (Emp, Nfunc, Prod)
{
    this.Emp = Emp
    this.Nfunc = Nfunc
    this.Prod = Prod

    /* retorna os dados em forma de uma string separada por virgula */
    this.toString = function() {
        return Emp+", "+Nfunc+", "+Prod;
    }
}
```

Podemos criar novas instâncias, usando a mesma estrutura, da seguinte forma:

```
Elogica = new Empresas("Elogica", "120", "Serviços")
```

```
Pitaco = new Empresas("Pitaco", "35", "Software")
```

```
Corisco = new Empresas("Corisco", "42", "Conectividade")
```

Assim, a variável Elogica.Nfunc terá o seu conteúdo igual a 120

Objeto Math

O objeto Math possui vários métodos que nós auxiliam nos mais diversos tipos de operação, este objeto não precisa ser estanciado, tendo que ser usado como está. Abaixo podemos ver os métodos que podem ser usados, assim como a descrição de suas operações.

Math.abs(número) - retorna o valor absoluto do número (ponto flutuante)

Math.ceil(número) - retorna o próximo valor inteiro maior que o número

Math.floor(número) - retorna o próximo valor inteiro menor que o número

Math.round(número) - retorna o valor inteiro, arredondado, do número

Math.pow(base, expoente) - retorna o cálculo do exponencial

Math.max(número1, número2) - retorna o maior número dos dois fornecidos

Math.min(número1, número2) - retorna o menor número dos dois fornecidos

Math.sqrt(número) - retorna a raiz quadrada do número

Math.SQRT2 - retorna a raiz quadrada de 2 (aproximadamente 1.414)

Math.SQRT_2 - retorna a raiz quadrada de 1/2 (aproximadamente 0.707)

Math.sin(número) - retorna o seno de um número (angulo em radianos)

Math.asin(número) - retorna o arco seno de um número (em radianos)

Math.cos(número) - retorna o cosseno de um número (angulo em radianos)

Math.acos(número) - retorna o arco cosseno de um número (em radianos)

Math.tan(número) - retorna a tangente de um número (angulo em radianos)

Math.atan(número) - retorna o arco tangente de um número (em radianos)

Math.pi retorna o valor de PI (aproximadamente 3.14159)

Math.log(número) - retorna o logaritmo de um número

Math.E - retorna a base dos logaritmos naturais (aproximadamente 2.718)

Math.LN2 - retorna o valor do logaritmo de 2 (aproximadamente 0.693)

Math.LOG2E - retorna a base do logaritmo de 2 (aproximadamente 1.442)

Math.LN10 retorna o valor do logaritmo de 10 (aproximadamente 2.302)

Math.LOG10E - retorna a base do logaritmo de 10 (aproximadamente 0.434)

Observação:

Em todas as funções, quando apresentamos a expressão "(número)", na verdade queremos nos referir a um argumento que será processado pela função e que poderá ser: um número, uma variável ou o conteúdo de um objeto (propriedade value).

Objeto Array

No Javascript podemos criar um objeto do tipo Array, capaz de se expandir cada vez que associamos um valor a uma posição do array ou inicializar o mesmo no momento de sua criação, assim reservando antecipadamente espaço para armazenar valores.

```
Ex.: MeuArray = new Array(); // Novo objeto Array vazio
      MeuArray = new Array(4); // Cria um vetor com quatro posições
      MeuArray = new Array(1, 2, "123", "AutoMAC"); // Cria um array com valores já definidos
```

Os arrays tem por padrão seu índice iniciado por zero, assim para acessarmos algum elemento devemos fazer uso de colchetes, por exemplo, MeuArray[0] irá retornar o primeiro elemento do array, os arrays podem ser usados como qualquer outro tipo de variável no Javascript, sendo ele uma variável que agrupa várias outras.

```
Ex.: MeuArray = new Array();
      MeuArray[0] = 1;
      MeuArray[1] = 3;
      var i = MeuArray[0]; // A variável i terá o valor 1;
```

Os arrays possuem várias operações de manuseio de seus elementos, abaixo poderemos ver os métodos e o que cada um deles pode fazer.

Array.join([separador]) - Forma uma string com o conteúdo dos elementos do vetor, sendo separado o caractere que os separa, se o mesmo não for passado, separação por virgula é adotada.

```
Ex.: var MeuArray = new Array("Elemento1", "Elemento2", "Elemento3");
      print(MeuArray.join("/")); // Forma a saída "Elemento1/Elemento2/Elemento3"
```

```
Ex2.: var MeuArray = new Array("Elemento1", "Elemento2", "Elemento3");
      print(MeuArray.join()); // Forma a saída "Elemento1,Elemento2,Elemento3"
```

Array.length – Indica o número de elementos do Array.

Valor = Array.pop() - Remove o ultimo elemento do Array e o retorna em Valor.

Array.push(valor1[, valor2, ...]) - Adiciona elementos ao final do vetor, este método pode receber uma parâmetro ou mais.

Array.reverse() - Reverte a posição dos elementos, no qual o primeiro elemento fica na última posição, o segundo sendo o penúltimo e assim por diante.

Valor = Array.shift() - Semelhante ao pop, mas este remove o primeiro elemento retornando em Valor.

NovoArray = Array.slice(inicio [, fim]) – Retorna um novo array formado pelos elementos definidos entre inicio e quantidade, fim deverá ser a quantidade de elementos desde o inicio do array, não podendo este ser menor que inicio.

```
Ex.: MeuArray = new Array(3, 5, 1, 0);
      NovoArray = MeuArray.slice(1, 3); // NovoArray possui os valores 5 e 1
```

NovoArray = Array.sort([function(elemento1, elemento2)]) – Ordena o Array conforme a função passada para sort, se está não for passada, ela irá retornar conforme ordem alfabética. A função a ser passada deverá retornar um valor conforme os elementos passados, este retorno deverá obedecer os seguintes critérios:

- se retornar valor < 0, elemento2 fica em índice menor que elemento1.
- se retornar valor = 0, elemento1 e elemento2 não mudam de índice.
- se retornar valor > 0, elemento2 fica em índice maior que elemento1.

```
Ex.: function compareNumbers ( a, b )
    {
        if ( a < b ) return -1;
        if ( a > b ) return 1;
        return 0; // a == b
    }
```

```
var ages = [ 7, 105, 14 ];
ages.sort(compareNumbers);
print(ages); // demonstra "7,14,105"
```

Array.splice(inicio, quantidade, [elemento1][, ..., elementoN]) – Este método pode inserir ou remover objetos, dependendo dos parâmetros passados, ou pode substituir elementos por outros. O seu uso se dá pela seguinte maneira:

inicio – quantidade de elementos que devemos ignorar ou pular.

quantidade – quantidade de elementos a serem removidos ou substituídos, se quantidade for igual a zero, devemos passar pelo menos um elemento a ser inserido.

elementoN – Elemento a ser inserido no local, podemos passar vários elementos por vez.

```
Ex.: num = new Array("10", "20", "30", "40");
      print(num); // escreve 10,20,30,40
      de_fora = num.splice(2, 0, "25");
      print(num); // escreve 10,20,25,30,40
```

Array.toString() - Retorna a string contendo todos os elementos do array, mesmo efeito da função join(), mas o separador é o default, no caso os elementos são separados por virgula.

Array.unshift(elemento1[, ..., elementoN]) – Semelhante ao push(), mas este adiciona elementos ao inicio do array.

Objeto String

O Javascript é bastante poderoso no manuseio de Strings, fornecendo ao programador uma total flexibilidade em seu manuseio. Abaixo apresentamos os métodos disponíveis para manuseio de strings.

String.length - retorna o tamanho da string (quantidade de caracteres)

String.charAt(*posição*) - retorna o carácter da posição especificada (inicia em 0)

String.indexOf("string") - retorna o número da posição onde começa a primeira "string"

String.lastIndexOf("string") - retorna o número da posição onde começa a última "string"

String.substring(*index1*, *index2*) - retorna o conteúdo da string que corresponde ao intervalo especificado. Começando no carácter posicionado em *index1* e terminando no carácter imediatamente anterior ao valor especificado em *index2*.

Ex.:

```

Todo = "Elogica"
Parte = Todo.substring(1, 4)
(A variável Parte receberá a palavra log)

```

String.toUpperCase() - Transforma o conteúdo da string para maiúsculo (Caixa Alta)

String.toLowerCase() - Transforma o conteúdo da string para minúsculo (Caixa Baixa)

String.escape ("string") - retorna o valor ASCII da string (vem precedido de %)

String.unescape("string") - retorna o carácter a partir de um valor ASCII (precedido de %)

Javascript no AutoMAC

No AutoMAC, podemos programar em Javascript no "Terminal de comandos" da aplicação, a única função exigida é a de nome start, que serve de ponto de partida para a configuração e inicialização de vários tipos de eventos, sendo ela aconselhada de uso para este fim. A seguir iremos ver os mais diversos tipos de controles e funções que podem ser manuseadas para controle das portas existentes no AutoMAC.

Objetos para controle de AutoMAC's

Para o controle dos eventos e ações que podem ser tomadas foram definidos alguns objetos e métodos que podem ser acessados para controlar AutoMAC's, a seguir você poderá visualizar uma lista desses objetos.

Objeto	Descrição	Exemplo
device	Controle geral de um AutoMAC, possui funções para controle de várias portas simultaneamente por exemplo.	device.io_restore(); /* Restaura todos as portas digitais para seu estado original */
io(0-13)	Controle de portas digitais.	io0.output(true); /* Saída digital 0 em nível alto */
da(0-1)	Controle de saídas analógicas	da0.output(2.0); /* Saída analógica 0 com saída em 2 Volts */
ad(0-7)	Controle de entradas analógicas	function ad0_receiver(value) { /* value possui o valor em volts de um dado */ print(value); } ad0.receiver(ad0_receiver);
ad_int		
timer	Controle dos temporizadores	timer.start(0, 1); /* Inicia timers 0 e 1. */
counter	Controle do contador de interrupções	counter.start(); /* Inicia contador */

Objeto *device*

Serve de controle geral de várias funções do sistema, nele podemos encontrar funções que executam ações específicas e que afetam como o AutoMAC irá funcionar.

Lista de Funções

device.pwm_period(*periodo*) - Período de funcionamento de um AutoMAC em microssegundos.

device.io_restore() - Restaura as portas digitais para seu estado inicial a qual se encontram todas com níveis de entrada.

device.ad_sample_rate(*amostragem*) – Atribui a amostragem dos portas analógicas, no caso a quantidade de amostras por segundo.

device.ad_receiver(*function(array)*) – Atribui o evento que deverá receber todas as amostras de dados AD simultaneamente.

Ex.:

```
function device_ad_receiver(ad_array) {  
    var i = 0;  
    for (; i < 7; i++)  
        if (ad_array["ad" + i] != undefined)  
            print("AD" + i + " " + ad_array["ad" + i]);  
}  
  
function start() {  
    device.ad_receiver(device_ad_receiver);  
}
```

Esse código ao ser ativado, irá mostrar os valores de amostras no canais AD ativados no momento.

device.ad_gain(*channel, gain*) – Atribui ganho no canal AD escolhido.

device.ad_fixed_conversion(*conversoes*) – Atribui taxa de conversões fixas para todos os canais AD.

Objeto *io(0-13)*

O objeto *io(0-13)* controla as interações das portas digitais, sendo que podemos enviar comandos de saída ou de entrada, dependendo do propósito do código que está sendo executado.

Lista de Funções

Considere N exposto aqui como se fosse um valor numérico de 0-13.

ioN.output(*true/false*) – Atribui a saída digital o valor passado a ela, podendo ser *true* ou *false*.

valor = ioN.input() - A variável *valor* possuirá o valor em *true* ou *false* do canal digital, dependendo do tipo de entrada aplicada. Esta função também serve para tornar uma porta digital em entrada.

ioN.pwm(*valor*) - Válido somente nos canais de 0-5, esta função atribui uma saída PWM em valor em escala de 0 a 100%.

ioN.is_output(*true/false*) -Verifica se o canal digital está atribuído como saída, sendo que pode ser verificado se está atribuído como saída com valor *true* ou *false*.

ioN.is_input() - Verifica se a porta digital está configurada como entrada.

ioN.is_pwm() – Válido somente nos canais de 0-5, esta função retorna se um canal digital está atribuído como saída PWM.

Objeto *da(0-1)*

As funções em *da(0-1)* controlam as saídas analógicas.

Lista de Funções

Considere N exposto aqui como se fosse um valor numérico de 0-1.

daN.output(*valor*) – sendo *valor* a saída em volts que podemos dar, sendo que ele poderá ter

valores de 0.0 e 3.3 volts.

Objeto *ad(0-7)* e *ad_int*

Este objeto serve para controle de leituras de portas AD, fazendo o controle de início e parada, além de recebimento de eventos.

Lista de Funções

Considere N exposto aqui como se fosse um valor numérico de 0-7.

adN.start() - Inicializa a análise de uma porta AD, gerando eventos que podem ser capturados `adN.receiver(function())` e `receiver_state()`.

adN.stop() - Para a análise de uma porta AD, parando a geração de eventos.

valor = adN.read() - Faz uma leitura da porta AD, recebendo a voltagem atual da porta.

adN.receiver(function(valor)) – Atribui a função que recebe o valor da voltagem decorrente da análise.

adN.receiver_state(function(estado)) – Recebe a função ou nome da função que recebe o estado atual da análise, se for true a análise foi iniciada e se false foi parada.

Objeto *timer*

O objeto timer controla os timers do sistema, recebendo eventos a cada timer configurado e disparado.

Lista de Funções

timer.start(timer[, ...]) – função que inicializa os timers conforme os parâmetros passados, disparando um evento ao final período do timer. No mínimo aceitando 1 parâmetro e no máximo até 4 deles.

Ex.: `timer.start(1, 2);` // Inicia timers 1, 2

timer.config(timer, ms) – Configura um timer individualmente, sendo o tempo de disparo aceito em milissegundos.

timer.receiver(function(timer)) – Atribui função que receberá o timer que foi disparado, sendo o parâmetro recebido o número do timer.

Objeto *counter*

Controla o receptor de interrupções digitais, servindo para configurar e inicializar, assim como receber eventos a cada interrupção atingida.

Lista de Funções

counter.start() - Inicializa o contador de interrupções, disparando um evento ao chegar o limite de contagem.

counter.pause() - Pausa o contador, sendo iniciado novamente com `start()`.

counter.stop() - Para o contador, assim parando a contagem e a geração de eventos.

valor = counter.read() - Lê a quantidade de contagens já feitas.

counter.config(borda = true/false, limite) – Configura a borda de subida (true) ou descida (false), e o limite de contagem.

counter.receiver(function()) - Configura a função que irá receber o evento de disparo de interrupção.