

# Criando Aventuras Para

# ABL

Adventure  
Basic  
Language

## Flight Simulator 2004



Por Ivan Sinigaglia Nunes Pereira

**D**esde a versão 2000 do Flight Simulator, a Microsoft disponibiliza uma série de aventuras, lições e provas de certificação junto ao pacote principal. São vôos que acompanham o manual do piloto para treinamento das lições e descrições do livro, vôos com orientação de voz e legendas, com manobras específicas e vôos de certificação, que disponibilizam certificados em níveis diferentes àqueles que conseguem vencer todas as restrições impostas pelo instrutor durante o vôo.

Muitas empresas têm utilizado esses recursos, desde versões anteriores ao Flight Simulator 2000, para criar aventuras de vôo, colocando nós, pilotos virtuais, a provas em situações específicas de acordo com a idéia e criatividade de cada um. Hoje em dia, as melhores aventuras comerciais podem ser encontradas na **Perfect Flights**, que tem uma coleção enorme de aventuras. Muitas vezes incluindo nos seus pacotes aeronaves e cenários. Já, gratuitamente, encontramos aventuras muito bem elaboradas na **FSAdventures**, uma



empresa australiana que disponibiliza uma quantidade grande de aventuras, um fórum de discussão de criação de aventuras e uma seção com softwares. Eles possuíam até um software comercial para edição das aventuras, quando a linguagem ainda não era ABL, que hoje está com distribuição gratuita.

Para entender um pouco para que realmente serve esse guia, vamos discutir um pouco sobre o que é a ABL e o que ela disponibiliza hoje. Esse guia foi criado, ou melhor, está sendo elaborado, da necessidade que tive de encontrar material sobre programação e criação de aventuras em português (que não existe) a fim de criar aventuras e lições em cenários brasileiros, na nossa casa, com vozes em português, com companhias aéreas nacionais, valorizando nosso mundo da simulação que é uma dos mais ricos no planeta. Por isso comecei a buscar tudo o que achei a respeito, desde o site oficial da Microsoft até desmembrando aventuras comerciais e gratuitas pra entender os códigos de programação.

A idéia deste trabalho é mostrar aspectos básicos de programação de uma maneira que mesmo quem nunca estudou programação



possa se aventurar a tentar criar suas próprias aventuras. Muito pouco, além do *Bloco de Notas* e muita vontade e paciência, vai ser preciso pra criar essas aventuras e lições. Você pode usá-las para divulgar seu cenário, para criar provas de admissão em sua companhia aérea virtual, simplesmente por criar aventuras, etc. Para a criação desse guia, ao contrário da idéia inicial de criar um pequeno guia e acrescentar a tradução do SDK FS2004 no final, já incorporei a tradução ao guia, criando um documento só: o **FS2004 SDK** (Software Development Kit — Advanced Script Language), como base, ao guia prático de utilização do ABL, com exemplos no final.

Boa leitura e bom trabalho.

Criando Aventuras Para Flight Simulator 2004

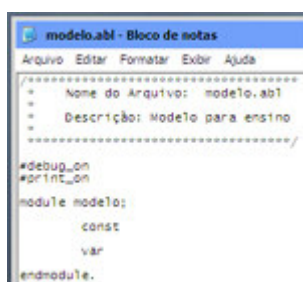
## A B L

## Adventure Basic Language

Antes de mais nada, vamos fazer um breve comentário sobre o que é a linguagem de programação ABL. Esta linguagem foi desenvolvida pela Microsoft para a criação e edição de aventuras e jogos para seus próprios softwares, incluindo o Flight Simulator. Ela é uma evolução da APL - Adventure Programming Language. Essa linguagem, utilizada até o FS2000, é uma linguagem onde era necessário uma compilação do script final. Isto é, você editava a aventura em um editor simples ASCII e depois um software fazia a compilação desse para um arquivo no formato utilizável de aventura que era **\*.ADV**. Esse arquivo, depois de pronto, ia para pasta [raiz]/adv de seu simulador para que essa aventura pudesse ser utilizada. O software que geralmente fazia essa conversão era o famoso **APLC32.EXE**.

Programas comerciais, como por exemplo o ProFlight 98/2000, já tinham incorporado no seu mecanismo de criação de aventuras esse compilador. Quem utilizou o ProFlight se lembra que no final, após estabelecidos todos os parâmetros de criação do voo, era feita a compilação final e criação da aventura. Isso não é mais necessário a partir do FS2002. Ainda é possível utilizar algumas dessas aventuras atualmente, mas particularmente acho que não vale a pena o trabalho. Até porque novos recursos foram incorporados com as aventuras novas, especialmente com o FS2004. Há poucos dias a Microsoft liberou a versão do SDK para o FS2004. Portanto, já estamos utilizando nesse guia os novos recursos de programação da linguagem. A Microsoft não atualizou a versão com os novos recursos mas nós aqui discutiremos o que tem de novo. Na verdade, o que mudou foi a incorporação de funções novas que permitem a manipulação principalmente de mensagens, pois os códigos são basicamente os mesmos. Até por isso acho que mantiverem o mesmo documento.

A linguagem ABL, ao contrário de sua antecessora, é uma linguagem de interpretação. Isto significa que você não vai precisar criar sua aventura, compilar, instalar e executar para testá-la. Não vai precisar também de módulos externos pra que elas funcionem como o mais conhecido - e atualmente pago - FSUIPC.DLL. Você cria e edita suas aventuras com a mais poderosa ferramenta do windows até hoje, o **Bloco de Notas**, e isso com o Flight Simulator em execução. E esse não precisa ser reiniciado para que as aventuras sejam reiniciadas ou novamente executadas. Erros gerados na leitura do script vão apenas gerar mensagens de erro e o voo será iniciado apenas sem o script da aventura. Em raros casos o sistema ficará instável ou travado a ponto de ter de ser encerrado. Atualmente, na verdade, utilizo um substituto gratuito para o bloco de notas que se chama EditPad Lite. Ele pode ser baixado na internet e tem a vantagem de ser um editor ASCII com vários recursos adicionais, principalmente com um contador de linhas. Você sabe, pela posição do cursor, que linha está escrevendo ou editando. Isso vale muito para a análise de erros. Quando é gerado um erro na execução do script, como veremos adiante, uma janela indica em qual linha está o problema. O bloco de notas nativo do Windows não tem esse recurso. Não utilize um editor de textos como o Word do Office porque estes editores geram caracteres que serão interpretados pelo programa como erros.



## Diretórios

Quando o MSFS é instalado, ele distribui seus arquivos por diferentes pastas. No FS2004 essa disposição mudou um pouco. Apesar disso, a tendência dos desenvolvedores de aventuras e lições é não seguir a rotina habitual do FS2004. Se você prestar atenção na sua pasta **Meus Documentos** você vai encontrar uma pasta **Flight Simulator Files**. Quando você cria um voo ou um plano de voo, normalmente é aí que o flight deposita os referidos arquivos. Você vai ter de recorrer à esta pasta quando criar seus voos para suas aventuras.

Sempre que você vir a referência [raiz] significa onde você instalou seu FS2004. Por exemplo, quando eu quiser me referir a pasta MODULES, vou me referir somente por *pasta MODULES* ou [raiz]/modules. No meu computador, o FS2004 está instalado em **c:\arquivos de programas\Microsoft games\Flight Simulator 9**. Portanto, este caminho todo aí atrás é a [raiz] (*em inglês [root]*) da instalação do meu simulador. Para que sua aventura apareça na janela principal dos voos quando você for selecionar um, Este deve estar colocado ou na pasta *meus documentos\flight simulator files* ou na pasta **[raiz]/flights**. A sugestão é inicialmente criar uma pasta em flights e usá-la para trabalhar com sua aventura, por exemplo, [raiz]/flights/modelo. Nesta pasta MODELO vamos construir nossa primeira aventura.

Outra pasta importante para a execução das aventuras é a pasta **[raiz]\scripts\lib**. Nesta pasta você vai encontrar uma série de arquivos com a extensão **\*.abl**. Esses arquivos fazem parte de uma biblioteca de variáveis, constantes e funções que são utilizadas por todas as aventuras e lições do FS2004. Você pode tanto utilizar os recursos dessa biblioteca como definir no seu próprio script mestre tudo o que tem nelas. Isso com o decorrer da leitura vai ficando mais claro. Não se preocupe em assimilar tudo isso agora. O importante é saber quais pastas são importantes e o que tem nelas.

Caso você se interesse em pesquisar o script de lições e aventuras já prontas, pra se especializar mais ainda no processo de criação de lições e aventuras, você vai encontrar esses scripts na pasta **[raiz]\lessons(...)**. Dentro de LESSONS você encontra ATP, COMM, INSTR, PRIV, SOLO e STUDENT. Dentro de cada uma você encontra vários scripts referentes as lições e as

provas de certificação (checkide).

Outra pasta importante com constantes que são utilizadas pelos scripts, e que é uma novidade do FS2004, é a pasta **[raiz]\messages(...)**. Também possui subpastas referentes a cada nível de pilotagem e também possui arquivos com extensão **.msg**. Nesses arquivos a Microsoft colocou as mensagens referentes a cada lição ou aventura, de forma a organizar melhor as mensagens para facilitar sua utilização. Uma única pasta em particular, **[raiz]\messages\tolerances**, tem um arquivo **tolerances.msg** com mensagens comuns a todas lições. Todos esses arquivos podem ser desnecessários quando você for criar suas aventuras e lições.

Por fim, os sons que são utilizados pelas lições e aventuras que já vem com o FS2004, arquivos com extensão **\*.wav**, ficam na pasta **[raiz]\sound\lessons**. Mais uma opção que pode ser mudado pelo desenvolvedor.



## Arquivos

Após termos vistos quais pastas nos servem na execução das aventuras vamos dar uma olhada

quais são os arquivos que nos interessam: (OBS: caso você não esteja conseguindo ver a extensão de seus arquivos, isso deve ser configurado no seu Windows Explorer em Ferramentas > Opções da pasta... > Modo de exibição > **desmarque** Ocultar as extensões dos tipos... na 17ª linha)

**ABL:** (extensão \*.abl):

Tanto o arquivo principal quanto a maioria dos satélites (onde ficarão funções ou outros tipos de instruções, caso necessário) tem essa extensão. Para dar uma olhada nele ou editá-lo, abra

com o Edit-Pad ou Bloco de Notas. Lembre-se

sempre de que ,ao salvar ou abrir, tenha optado em *Arquivos do tipo:* escolhido a opção **Todos**. É óbvio que para a maioria isso parecer desnecessário, mas para alguns não. Como a idéia desse guia é oferecer orientação básica e inicial, vamos discutir desde o início.

**Vôo:** (extensão \*.flt): Esses arquivos já são mais conhecidos dos usuários do flight e contém as informações do voo. Um voo deverá inicialmente ser criado (não um plano de voo): escolha o tipo de aeronave, local de partida, condições climáticas e horários. Esse arquivo também é editável pelo EditPad e nele devermos fazer a referência para qual script usaremos. Veremos isso mais adiante.

**Clima:** (extensão \*.WX): Aqui ficam armazenadas as informações sobre as condições climáticas que você definiu quando criou seu voo inicial. Pode ser modificado após criado.

**Resumo:** (extensão \*.htm): (*Flight Briefing*). Neste arquivo ficam as informações que aparecem antes da aventura, em Flight Briefing, ou lição ser iniciada, contendo informações e instruções sobre o voo que será executado. É um arquivo escrito em HTML que pode ser editado por muitos programas, por exemplo, o Microsoft Front Page e o próprio EditPro, da mesma maneira que é criada uma página para a internet. Pra dizer a verdade, é uma página de internet que é inserida nesta área do simulador com a possibilidade de ser impressa. Podem ser utilizadas imagens e essas devem ficar na mesma pasta que o arquivo HTML. Ela deve ter o mesmo nome de arquivo antes de \*.htm do arquivo principal (ABL) para que possa ser lido ao se iniciar a lição.

Outro arquivo que pode ser incluído na pasta onde os arquivos serão instalados é o **desc.txt**. Nesse arquivo ficarão algumas informações sobre a aventura que vão aparecer na tela de *Select a Flight*. Você mesmo pode criá-lo e inclua as seguintes linhas (ou melhor, ele só terá essas 3 linhas):

[MAIN]

Title= Aventura 01

Description= Modelo de teste

Ai salve o arquivo como desc.txt e pronto.

**Arquivos de som:** (extensão \*.wav): Esses são os arquivos com os diálogos e/ou legendas que aparecem no flight. Você pode criar arquivos de som novos ou utilizar arquivos já existentes. O formato recomendado é 22,050 samples por segundo, PCM adaptativo (ADPCM.) Caso arquivos de som de diferentes formatos ou taxas de amostragem forem tocados conjuntamente, a qualidade do som final vai sofrer. Uma exceção pode ser feita para arquivos de comprimento-zero (*zero-length files*) que estão presentes somente para as legendas. Se há texto na porção *INAM* do arquivo wave, este será mostrado no topo da tela à medida que o arquivo wave é tocado. São aquelas informações de

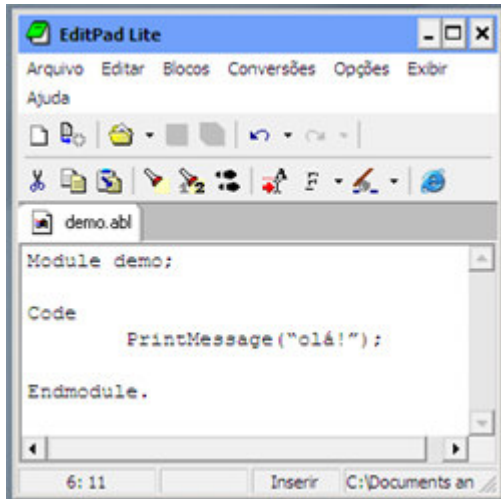


texto que aparecem na parte superior da tela, em uma faixa verde. Para editar esses arquivos, você vai precisar de um programa que faça isso. O mais recomendado é o Adobe Audition, que é a versão da Adobe para o CoolEdit, que ela adquiriu. O software é o mesmo. Como fazer isso veremos mais adiante.

Com exceção dos arquivos de som, todos os outros devem ficar na mesma pasta. Crie uma pasta em [raiz]\flights\ com o nome de MODELO, que usaremos para dar continuidade na criação e edição da aventura de demonstração.

## O SCRIPT

Um script em sua forma mais simples contém uma declaração **MODULE** seguida pelo nome do módulo e **ponto-e-vírgula (;)**; o **Code** (o código propriamente dito), com uma instrução de impressão nele, e a declaração **ENDMODULE** no final, seguida por **ponto final**. Esse exemplo acima, que está na figura a seguir, é o básico do básico.



Aliás, o exemplo acima não vai surtir efeito algum porque faltam ainda outras instruções. Cada declaração foi colocada em uma linha e o número de linhas em branco entre elas não tem restrições. Crie o arquivo acima **demo.abl** e salve-o na pasta [raiz]\Flights\Modelo. As declarações acima são as únicas obrigatórias no script. Para quem já programa em alguma linguagem, não vai ter dificuldades com a ABL. Ela se parece um pouco com PASCAL mas não é de nenhuma outra linguagem; ABL é apenas ABL.

Além das declarações **MODULE** e **ENDMODULE**, delimitando o “espaço físico” do script, temos as seções **Code**, **Const** e **Var**, que serão descritas a seguir. Dentro do script teremos ainda as funções (**function**), que trazem instruções que normalmente se repetem pelo código. Além disso, é sempre de bom tom, fazer anotações dentro do script para se saber pra que servem determinadas declarações, constantes, variáveis e funções. Em ABL você pode fazer isso de duas maneiras:

1) No início da linha você pode usar um duplo “//” seguido pelo texto que você quer escrever. Esse “//” faz com que essa linha não seja lida pelo código. Pode escrever quantos caracteres quiser a qualquer distância dele que não tem problema. Só não pode mudar de linha. Esse sinal é para cada linha, se você quiser escrever a instrução em mais de uma linha, use // para cada linha;

**Exemplo:**

// Aqui ficarão as variáveis do código

2) Colocando todo o texto que você quer que não entre como parte do código entre /\* e \*/.

**Exemplo:**

/\* Aqui ficarão as informações do código que você quer ocultar. Essas podem aparecer em múltiplas linhas desde que você não esqueça de fechar o conteúdo \*/

```
Module demo;

#print_on
#debug_on

Const
    // Constantes
    nVNE10      = 250;
    nVMO        = 490;
    nMMO        = 0.740;
    nVLRC       = 447;
    mensag01    = "Instrução Inicial.";
    V1          = 130;
    V2          = 140;

Var
    // Principais Variáveis
    Static World      wTheWorld;
    Static AutoPilot  AP;
    Static Aircraft   acMe;
    Static Cockpit     cpOffice;
    Static Shell       S;
    Static Flight      fltThisFlight;
                   Altitude  nAltitude

//Função para tocar arquivo de som
function fnSQ_Play(String Filename);
var
    String strFileSpec;

code
    strFileSpec = fn3StrBuild(strWavBase,Filename,".WAV");
    fnSQ_Private(SQ_Play,strFileSpec,0,0,FALSE);
endfunction;
//Fim da função

// Abaixo é o código propriamente dito.
Code
    PrintMessage("olá!");
Endmodule.
```

Acima, coloquei um exemplo de script somente com instruções ilustrativas; não vai funcionar. É apenas para podemos descrever o que teremos nos scripts que vamos programar e encontrar por aí. A maioria vai ter essa cara, provavelmente com muito mais linhas. Agora, além das declarações descritas anteriormente, vemos aqui novas declarações: *Const*, *Var* e *function*.

**Const** (Constantes): Atribui nomes amigáveis a constantes numéricas ou de caracteres, aprimorando a legibilidade e sustentação do script<sup>1</sup>.

Como o próprio nome diz, constantes são constantes, isto é, não variam; são fixas. Se variassem, seriam variáveis (durghhh....). Mas a idéia é essa mesmo. Você atribui na seção constantes valores numéricos ou literais a nomes que utilizará depois na seção code e nas funções. No script acima temos, por exemplo, **V1=130**; Isso significa que V1 sempre terá o valor nessa aventura de 130. Também **mensag01="Instrução Inicial."**; sempre que for solicitada mensag01 essa retornará a mensagem *Instrução Inicial*.

Se você der uma olhada em [raiz]\scripts\lib vai encontrar um arquivo chamado **stdkonstants.abl**. Nesse arquivo você vai encontrar várias constantes globais para todas as lições e aventuras. Você não precisa incluí-lo no seu script mas é bom dar uma olhada para se familiarizar. Vários outros scripts incluem suas próprias constantes independente das globais.

Você pode tanto criar um arquivo que contenha suas constantes, e chamá-lo de seu script principal, quanto incluí-las no próprio script principal. Não esqueça de deixar anotações pra lembrar pra que servem determinadas constantes.

**Var** (Variáveis): Uma variável é declarada especificando-se um nome, dimensões de arranjo se a variável é um arranjo (*array*) e um ou mais nomes de variáveis delimitadas por vírgulas. Arranjos de até oito elementos podem ser declarados. Cada frase de declaração de variável deve ser terminada com ponto-e-vírgula e cada variável tem um tipo e uma classe de armazenamento <sup>1</sup>.

Seguindo a mesma lógica das constantes, aqui vamos declarar as atribuições variáveis que utilizamos com os vãos. Por exemplo, a altitude da aeronave varia durante o trajeto da decolagem ao pouso. Ela varia de acordo com a distância e local. A não ser que você crie uma aventura que se inicie em uma determinada altitude e essa fique constante até o final - para avaliar determinadas manobras sem que a altitude varie - ela vai variar, portanto você deve definir uma variável para trabalhar com a altitude.

ABL suporta dois tipos nativos de dados para as variáveis: *números* e *caracteres*. Além dos tipos nativos, a implementação do ABL no Flight Simulator inclui uma variedade de definições de tipos de objetos, que veremos mais adiante. Os **tipos** nativos são:

### NUMBER

(número): É usado para representar todas as quantidades numéricas e Booleans. Exemplos de valores numéricos válidos incluem: TRUE (verdadeiro), FALSE (falso), 10, e 109.3153. É equivalente ao que temos na linguagem C. É garantida a precisão de até 15 dígitos e representa uma magnitude entre  $10^{308}$  e  $10^{-308}$ .

### STRING

(caractere): é o equivalente de um arranjo de caracteres C terminado-nulo e pode ter um tamanho até de 1024 caracteres. Exemplos de strings incluem "E aí, beleza?", "10.531" e "". São as "palavras" e símbolos de uma maneira geral. Os números aqui não têm "valores numéricos", representam apenas resultados ou valores para serem exibidos. Não se faz contas com funções matemáticas com strings, somente se esses números de strings forem convertidos para number.

Além dos tipos, as variáveis possuem diferentes **classes de armazenamento**. *Classes de Armazenamento (Storage class)* determinam se o valor da variável é perdido ou retido entre as chamadas de scripts e se o valor da variável será visível em outros scripts.

As três classes de armazenamento suportadas pelo ABL são: **static**, **eternal** e **automatic** (*estática*, *permanente* e *automática*). Se o tipo declarado de uma variável é precedido pela palavra "static", é uma variável estática. Se é precedida por "eternal", é uma variável permanente. Caso contrário, é uma variável automática.

**Variáveis estáticas (static)** são alocadas quando o módulo é carregado e persistem até que o módulo seja descarregado, indiferentemente de quantas vezes o script seja executado. Uma variável estática é visível somente no módulo ou função na qual é declarado.

**Variáveis permanentes (eternal)** são um tipo especial de variável estática. Estas são trocadas entre os módulos. Uma vez tendo a ABL carregado um módulo definindo um valor a uma variável como permanente, esta variável será visível em todo lugar em todos os módulos.

**Variáveis automáticas** são alocadas temporariamente no ABL todas as vezes que vierem no escopo e estas serão liberadas quando este terminar. Cada vez que forem liberadas, todo o seu valor será perdido. Uma variável automática é visível somente no módulo ou função na qual é declarada.

Ainda não chegamos ao estudo das funções mas já adiantando, veremos algumas regras que servem para elas e os módulos. Dentro dos módulos teremos funções, que executam ações específicas várias vezes no código. Elas são mais ou menos "mini-módulos" que possuem seções de variáveis, constantes e código dentro dessas. As variáveis das funções se relacionam da seguinte maneira com as variáveis do código principal:

As variáveis declaradas na seção *Var* do módulo serão vistas em todas as funções declaradas no módulo. Se a função no módulo declara uma variável com o mesmo nome de uma variável do módulo principal, a função perde acesso à variável do módulo, isto é, fica valendo a variável da função enquanto essa estiver sendo executada. Variáveis de função se sobrepõem às variáveis do módulo.

Variáveis declaradas em uma função não são visíveis em outras funções ou na seção *Code* do módulo.

Variáveis permanentes devem ser declaradas somente na seção *Var* do módulo.

"Por questões ligadas à usabilidade dos termos no processo de programação e facilitar a compreensão aqui nesse guia, vamos sempre usar para as expressões numéricas e para as variáveis tipo número sua denominação original em inglês **number**, e para as expressões de caracteres e variáveis de caracteres, **string**."

Dê uma olhada também no **Apêndice 01**, no final desse guia, em **Notação Húngara e Nomes de Variáveis**.

Exemplos de formas de declaração de variáveis

```
Const
    max    = 10;

Var
    number i, j, k;

// Arranjo de 20 números, MyArray[0]..MyArray[19]
    number [20] MyArray;

    number [5,3,10,max]
    MyOtherArray;
    Static string    s, t;
    Eternal number q;
```



**Code (Código):** Esta seção contém todas as **declarações** executáveis para um módulo ou função e contém zero ou mais declarações. Cada **declaração** é terminada por ponto-e-vírgula e pode ser considerada como uma operação simples executada por um script. Na seção *Code*, a execução se processa de cima para baixo. Quando o fim é atingido ou quando uma declaração de retorno – **RETURN** – é encontrada, a execução se completa. Caso a seção *Code* seja uma função, a execução continua com a linha seguindo o ponto onde a função foi chamada. Caso contrário, a execução para, e o ABL retorna o controle ao Flight Simulator.

Em *Code*, temos quatro classes de declarações:

1. **Atribuição:** Define o valor da variável. Uma declaração de atribuição inclui: uma **variável** para receber um novo valor, um sinal de equivalência e uma **expressão** para atribuir uma variável. Há três tipos de expressões e variáveis: **number**, **string** e **object**.

`A = 1;            nIASNow = 140;`

## 1a. Number:

**Variáveis Numéricas:** recebem o valor de expressões aritméticas. Uma expressão aritmética pode ser muito simples, como uma literal numérica, variável numérica ou uma função que retorna um número. Expressões aritméticas podem incluir expressões matemáticas envolvendo muitas expressões aritméticas simples.

### Expressões Aritméticas

Uma expressão aritmética consiste de um ou mais valores numéricos, separados por operadores aritméticos, relacionais ou lógicos. Valores numéricos incluem:

**Um número literal:** Por exemplo, 10.5, 3001.51

**Um número constante:** Este é uma constante declarada na seção *Const* associada a um literal numérico.

**Um número variável:** Este é uma variável declarada do tipo *number*, o qual foi iniciado antecipadamente.

**Um elemento simples num arranjo de números:** Cada elemento individual no arranjo é um número. Elementos de arranjo são indexados de 0 a dimensão-1.

**Uma função que retorna um número:** Por exemplo, a função *sqr(n)* retorna a raiz quadrada (*square root*) de seu parâmetro *n*.

**Uma propriedade de objeto do tipo numérico:** Por exemplo, se a variável de um objeto do tipo *Aircraft* foi declarada, *TrueAirspeed* é uma propriedade daquele objeto, que é um número – *trueairspeed* (*velocidade real no ar*) do *aircraft* (*aeronave*) em *knots* (*nós*).

**Operadores Aritméticos** válidos, em ordem de precedência são: (NT: regras computacionais definindo a ordem na qual as operações matemáticas são calculadas. Normalmente as multiplicações são executadas primeiro, depois divisões, adições e as subtrações por último),

**Parênteses:** (e) – parênteses agrupam operações, forçando-as a serem resolvidas antes que qualquer outra operação seja aplicada.

**Multiplicação e divisão:** \* e /; *a\*b* gera o produto de *a* e *b*; *a / b* retorna o quociente da divisão de *a* por *b*.

**Adição e subtração:** + e -. Estes operadores podem também ser usados como operadores unários (*unary operators* ?), assim como (-*a*).

**Operadores Relacionais** têm uma precedência menor do que os operadores aritméticos. Quando dois números são comparados por operadores relacionais, o resultado é **TRUE** (verdadeiro: qualquer número *exceto 0 – zero*, geralmente 1) ou **FALSE** (falso: 0 – zero). Os verdadeiros são:

**Menor que:** < (*a < b*) avalia como TRUE se *a* for menor do que *b*; caso contrário, avalia a sentença como FALSE.

**Igual:** = (*a == b*) avalia como TRUE se *a* for o mesmo que *b*; caso contrário, avalia a sentença como FALSE.

**Maior que:** > (*a > b*) avalia como TRUE se *a* for maior do que *b*; caso contrário, avalia a sentença como FALSE.

**Menor ou igual que:** <= (*a <= b*) avalia como TRUE se *a* for menor ou igual do que *b*; caso contrário, avalia a sentença como FALSE.

**Maior ou igual que:** >= (*a >= b*) avalia como TRUE se *a* for maior ou igual que *b*; caso contrário, avalia a sentença como FALSE.

**Não igual:** <> (*a <> b*) avalia como TRUE se *a* não for igual a *b*; caso contrário, avalia a sentença como FALSE.

**Operadores Lógicos** são de menor precedência do que ambos aritméticos e relacionais. Eles permitem somente variáveis únicas ou expressões entre parênteses como seus operandos. Há três operadores relacionais:

**NOT Negação:** (*NOT a*) avalia como TRUE se *a* é FALSE, caso contrário, avalia como FALSE.

**AND Conjunção:** (*a AND b*) avalia como TRUE se *a* não for FALSE **E** *b* não for FALSE. Em outras palavras, ambos *a* e *b* devem ser não-zero para a expressão ser verdadeira.

**OR Disjunção:** (*a OR b*) avalia como TRUE se *a* não for FALSE **OU** *b* não for FALSE. Em outras palavras, se *a* é não-zero ou *b* é não-zero ou se ambos forem não-zero, a expressão é TRUE.

**Operadores aritméticos** trabalham apenas como tipos numéricos. **Operadores lógicos e relacionais** sempre trabalham com tipos numéricos e às vezes com tipos de objetos (*object*) e de caracteres (*string*).

### Exemplos:

```
Const
    Ten      = 10;
Var
    Number a, b, c, d;
Code
    A = TEN + 15;      // A = 25
    B = (A - 5) / (5 * 2) + 2; // B = 4
    C = A and (b <= 4); // C = 1 (true)
    D = C and false;   // D = 0 (false)
    D = C or false;    // D = 1 (true)
```

## 1b. String:

**Variáveis de Caracteres:** recebe o valor de uma expressão de caractere. Esse valor é geralmente uma string literal ou o valor de uma função que retorna uma string. Ainda há uma função especial chamada *concat()* a qual concatena (*liga, encadeia*) o valor de uma string com a outra.

Strings **não** podem incluir operadores aritméticos, lógicos e relacionais. As únicas expressões válidas são atribuições de uma outra string, uma string literal ou uma função que retorne uma string.

O uso do caractere “\” nas strings segue a convenção da codificação da linguagem C. Por exemplo, na declaração **PRINT()**, habilitada pela diretiva<sup>2</sup> **#PRINT\_ON**, você pode incluir quebras de linhas na execução do aplicativo digitando:

**Print(“line one \n line two”)**

Similarmente, se a string for tratada como *caminho* (assim como quando usamos **#INCLUDE** para bibliotecas), você deverá usar “\\” no lugar de “/” (slash).

**#include “lib\\mylib.abl”**

### Exemplos:

```
Const
    Me = "John";
Var
    String    s, t, u, v;
Code
    S = Me;           // S = "John"
    S = "You";        // S = "You"

    // Concat é uma função que une a
    // string ou uma string contendo um
    // número no segundo argumento à
    // string no primeiro argumento

    concat(S, 10);
    // S = "You10"
```

## 1c. Object:

**Variáveis de Objeto:** Recebe o valor de uma expressão de objeto. Há muitos diferentes tipos de objetos, tipo *Aircraft* ou *Engines*. Uma variável de objeto pode apenas receber o valor do mesmo tipo de objeto ou um objeto similar. Essas expressões vêm em duas categorias:

*objeto aberto (naked object)* e objeto de propriedade de acesso (*object member property access*).

Um objeto aberto é um objeto usado numa expressão que tem propriedades próprias, mas das quais

não são acessadas na expressão. Um objeto de propriedade de um outro objeto, assim como *A.Autopilot* no exemplo usado anteriormente, pode ser considerado um objeto aberto. Da mesma forma, chamadas de funções assim como **GetWorld** (o qual retorna um objeto *World*), podem retornar um objeto aberto. Com relação ao entendimento dos objetos, teremos exemplos durante o desenrolar desse guia para uma melhor compreensão. Só para dar uma idéia, é através dos objetos que controlaremos a maioria das atividades que queremos que apareça em nossas aventuras.

Existe uma hierarquia na classe dos objetos. Por exemplo, para você determinar a altitude que estará sendo ajustada no piloto automático da aeronave em determinado momento, devemos seguir a hierarquia: *.World.Aircraft.Autopilot.Altitude* (a altitude do piloto automático da aeronave do mundo atual). Mas veremos isso com mais detalhes mais adiante.

Um objeto de propriedade de acesso pode evoluir para um objeto aberto ou um objeto terminal. Uma propriedade terminal (*terminal property*) é uma propriedade que não pode ser seguida por um ponto final (.). Deve ser do tipo number ou string ou ser uma função individual.

Expressões de objeto envolvendo uma propriedade terminal são classificadas como do tipo propriedade terminal. Por exemplo, *A.Autopilot.Altitude* é do tipo number, (retorna um número que será a altitude no piloto automático) desde que a propriedade *Altitude*, propriedade do objeto *Autopilot*, seja do tipo number. Essa expressão poderia ser usada em

qualquer lugar em que uma expressão do tipo number seja permitida.

A maioria das expressões de objeto são apenas de leitura (read-only) e isso significa que a propriedade objeto pode ser usada apenas em expressões; não podem aparecer como valor da esquerda nas instruções de atribuição.

### Exemplos:

```
Var
    Aircraft      A, B;
    Autopilot     AP;
    World         W;
    Number        n;
    Vector        v;
Code
    W = GetWorld;
    // Atribuição/expressão de objeto
    // aberto.

    W.PrintMessage("E ai, tudo bem?");
    // Propriedade de acesso individual
    //(função) sem valor de retorno

    A = W.GetAircraftById(0);
    // Atribuição aberta e propriedade
    // de acesso (função) com valor de
    // retorno o objeto Aircraft.

    B = A;
    // Atribuição aberta

    n = A.TrueAirspeed;
    // Objeto individual de propriedade
    // de acesso – variável individual

    AP = A.Autopilot;
    // variável individual E objeto aberto`

    n = A.Autopilot.Altitude;
    // Objetos múltiplos de propriedade
    // de acesso
```



2. **Seleção:** Decide se e qual sentença será executada. Por exemplo, escolher se vai subir ou descer a aeronave após determinado período de tempo. A construção **IF / ELSE / ENDIF** é uma típica declaração de seleção.

```
If (nIASNow >= 200) then
    Cpoffice.Flaps = 1;
endif;
```

Os mais comuns

tipos de seleção são as expressões **IF / ELSE / ENDIF**. Outro tipo comum de seleção é a expressão **SWITCH**, que permite que o código escolha uma larga possibilidade de instruções a executar baseada num simples valor numérico de entrada. Ambos esses mecanismos de seleção podem ser usados com ABL.

### O seletor IF

O seletor **IF** é também conhecido como uma instrução

#### IF / ELSE / ENDIF.

Uma instrução **IF** consiste de um identificador **IF** seguido por uma instrução numérica entre parênteses, seguida pela palavra **THEN**.

Se a instrução entre parênteses for **TRUE** (i.e., é um número e é não-zero), a instrução imediatamente seguida por **THEN** será executada até atingir **ELSE** ou **ENDIF**. Ai a instrução pula para a próxima expressão após **ENDIF**.

A cláusula **ELSE**, caso presente, vem após a sentença a ser executada na instrução se esta for **TRUE**, mas antes da expressão **ENDIF**. Se há a cláusula **ELSE** e a instrução na sentença **IF** é avaliada como **FALSE**, a sentença entre **IF** e **ELSE** será pulada e a sentença entre **ELSE** e **ENDIF** será executada.

As sentenças com **IF / ELSE / ENDIF** podem incluir sub-rotinas com sentenças **IF**. Tenha certeza de que sentenças **IF**, **ELSE**, e **ENDIF** estão apropriadamente inseridas.

```
if (nIASNow >= 200) then
    cpOffice.FlapsHandlePos = 0;
else
    cpOffice.FlapsHandlePos = 1;
endif;
```

Caso (if=se,caso) a velocidade no ar da aeronave ( que foi definida pela variável *nIASNow*) seja igual ou maior do que 200 nós, então (then=então) os flaps devem ser recolhidos (definidos pela posição 0); caso contrário (ELSE), ficam na posição 1 (que equivale no 737 a 1°).

*nIASNow* foi definida na seção *Var* como:

Number *nIASNow*; // Velocidade em IAS atual

**Depois iniciada em Code** como:

```
wTheWorld= getWorld(); // Objeto World
acMe = wTheWorld.AircraftByID(0); // Aeronave
nIASNow = acMe.IndicatedAirspeed; // Velocidade
```

Não estresse. É somente um exemplo.

### O seletor SWITCH

Muitos operadores de seleção envolvem a avaliação de uma expressão aritmética simples e depois a escolha de diversas expressões para execução baseadas no valor da expressão. Isto se torna muito difícil com uma variedade grande de expressões quando se usa **IF**. O seletor **SWITCH** permite a utilização desta operação de uma forma simples e bem organizada. Praticamente todas as lições e aventuras que vêm originalmente no Flight Simulator utilizam **SWITCH/CASE** como base de funcionamento no código.

O seletor **SWITCH** tem dois componentes: a expressão **SWITCH** e as expressões individuais **CASE** dentro dela:

1. A expressão **SWITCH** consiste do identificador **SWITCH** seguida por uma expressão integral entre parênteses. A instrução continua abaixo de **SWITCH** até que seja fechada ao encontrar a expressão **ENDSWITCH**.

2. A expressão **CASE** consiste da palavra **CASE** seguida por uma literal ou constante numérica e depois por dois pontos (:). Ai segue-se zero ou mais expressões, concluídas pela expressão **ENDSWITCH**. Se a expressão numérica da instrução **SWITCH** avalia o número dado numa instrução **CASE**, as expressões dentro do **CASE** são avaliadas até que uma expressão **ENDCASE** seja encontrada. Ai a execução salta para a instrução seguinte a instrução **ENDSWITCH**. Caso nenhuma instrução **CASE** iguale o valor da expressão na instrução **SWITCH**, a execução salta para a próxima instrução após a instrução **ENDSWITCH**.

Duas instruções **CASE** dentro de uma operação **SWITCH** não podem ser associadas com a mesma constante.

Const

```
STATE_STOPPED = 1;
// Estado da aeronave: parada.

STATE_TAXIING = 2;
// Estado da aeronave: taxiando.

STATE_AIRBORNE = 3;
// Estado da aeronave: voando.
```

Var

```
Number MyState, i, j;
String Message;
```

Code

```
// Chama a função GetPlaneState , a qual
// retorna um inteiro e não tem parâmetros
```

```
MyState = GetPlaneState;
```

```
// Se MyState não for equivalente a um dos
// três casos, nenhum código será usado.
```

**switch** (MyState)

```
case STATE_STOPPED:
    Message = "Você está em solo";
    i = 21;
endcase;
```

```
case STATE_TAXIING:
    Message = "Você está taxiando";
endcase;
```

```
case STATE_AIRBORNE:
    Message = "Você está no ar";
endcase;
```

```
case 4:
    Message = "Ei, o que é isso?"
```

**endswitch;**

```
// Exibe o resultado final
PrintMessage(Message);
```





3. **Iteração:** Executa uma ou mais declarações muitas vezes até que uma condição específica seja satisfeita ou até que o programa seja concluído, também conhecida como "looping". Típicamente usadas para processar informações em listas de arranjos.
- ```
For j = 1 to (i - 2) do
  k = k * j;
Endfor;
```

As instruções de iteração (*repetição*) permitem ao programa executar um bloco de código zero ou mais vezes. Também são conhecidas como *looping* e a expressão responsável por causar a repetição (iteração) pode ser chamada de *loop* (*repetidor*).

ABL suporta duas categorias de repetição: **pré-teste** e **pós-teste**.

Uma repetição **pré-teste** avalia uma expressão numérica antes de se executar qualquer loop no código. Se a expressão é **FALSE**, o loop não é executado e a execução continua imediatamente após o fim do loop. Caso contrário, o código no loop é executado até o fim do loop ser atingido. Depois, a execução retorna ao topo do loop, no seu teste. As duas repetições pré-teste são **WHILE / ENDWHILE** e **FOR/ENDFOR**.

Uma repetição **pós-teste** executa o conteúdo do loop e depois testa para ver se este poderá ser executado novamente. Caso negativo, a execução continua abaixo do teste. Se outra repetição é requisitada, a execução continua com a primeira instrução no loop. A repetição pós-teste é **REPEAT/UNTIL**.

### WHILE / ENDWHILE

A instrução **WHILE** (*ing: enquanto*) consiste do identificador **WHILE** seguido por uma expressão numérica entre parênteses e a palavra **DO** (*ing: faça*). Cada sentença que vai de **WHILE** até **ENDWHILE** (requerida pra fechar o loop) é considerada parte do loop. A expressão **WHILE** é avaliada antes de cada repetição. Se a instrução for **FALSE** o loop termina e a próxima instrução a ser executada é a que vem a seguir de **ENDWHILE**. Se a instrução não é **FALSE**, as instruções entre **WHILE** e **ENDWHILE** serão executadas sequencialmente. Quando a instrução **ENDWHILE** for encontrada, a execução pula de volta para **WHILE** no início do loop, começando este novamente.

Ao escrever um loop com **WHILE**, tenha certeza de que a expressão na instrução **WHILE** irá de

alguma maneira ao final avaliar para **FALSE** a fim de que o loop termine.

### FOR / ENDFOR

O loop **FOR** é um loop pré-teste que tem três tipos de operações de repetição: *inicialização*, *teste* e *incremento*.

A instrução **FOR** consiste da palavra-chave **FOR** seguida por uma variável, um sinal de equivalência, uma expressão numérica "inicial", a palavra **TO**, uma expressão numérica "final" e a palavra **DO**. Por exemplo:

```
For i = 1 to (k + 1) do
```

O loop contém todas as instruções a seguir de **FOR** até que uma instrução **ENDFOR** seja encontrada. Antes do loop ser testado ou executado, a variável do loop é ajustada à condição inicial, a expressão entre = e a palavra **TO**. No exemplo acima, a inicialização ajusta a variável **i** para 1.

Para cada repetição, o valor da variável de repetição é comparado à condição de finalização especificada entre **TO** e **DO** (**k+1**, neste exemplo). Se o valor de repetição da variável for maior do que a condição de finalização, o loop não será executado e a execução saltará para a próxima instrução após **ENDFOR**. Se o valor da variável for igual ou menor do que a condição de finalização, a instrução imediatamente após **FOR** será executada. Quando o **ENDFOR** correspondente for encontrado, a variável é aumentada em 1 e a execução retorna à instrução **FOR**. Quando o loop completar, o valor da variável deve ser ajustado à condição de finalização + 1. No exemplo acima, **i** deverá ser **k + 2**.

### REPEAT/UNTIL

A repetição **REPEAT / UNTIL** é uma repetição pós-teste. Ela consiste de uma palavra-chave **REPEAT** seguida por um bloco de um ou mais instruções, terminada por uma instrução **UNTIL**. A instrução **UNTIL** consiste da palavra **UNTIL** seguida por uma expressão numérica entre parênteses terminada por ponto-e-vírgula.

A repetição **REPEAT / UNTIL** executa o bloco de código entre **REPEAT** e **UNTIL** (*ing: até que*). Quando atinge a instrução **UNTIL**, esta avalia a expressão entre parênteses. Se a expressão for **FALSE**, a execução salta para a instrução **REPEAT** e o loop é executado novamente. Se a expressão é **TRUE**, o loop termina e a execução continua com a instrução a seguir de **UNTIL**.

Assim como a instrução de loop **WHILE**, você deve tomar cuidado para ter certeza de que o loop de alguma maneira terminará. Em outras palavras, assegure-se de que a expressão **UNTIL** eventualmente avaliará como **TRUE**.

```
Var
  Number i, j, k;
  Number f;

Code
  f = 17.5;
  i = 0;

  // Este loop repete até que f <= 1.0,
  // o qual deverá resultar em seis
  // repetições (f = 17.5, 8.75, 4.375,
  // 2.18..., 1.09..., 0.54...)

  While (f > 1.0) do
    MyFunction(f);
    i = i + 1;
    f = f / 2.0;
  Endwhile;

  k = 1;
  // 'i' foi definido acima em 6,
  // portanto essa repetição loop
  // será repetida quarta vez (j=1,2,3,4)

  For j = 1 to (i - 2) do
    k = k * j;
  Endfor;

  // Este loop irá executar apenas
  // uma vez, desde que o loop
  // anterior ajuste k
  // para 1*2*3*4 = 24.

  Repeat
    k = k - 1;
  Until (k < 100);

  // Loops em loops. 'k' recebe o valor
  // de 1 a 50 na sequência

  for i = 1 to 10 do
    for j = 1 to 5 do
      k = j + ((i - 1) * 5);
    endfor;
  endfor;
```

4. **Funções:** Uma declaração consistindo de somente uma função de chamada. Chamando a função, essa executa a declaração na seção de funções do *Code*.

Funções têm seus nomes de origem na matemática. Na matemática, uma função é uma fórmula feita para receber variáveis – “parâmetros” – que resultam em um valor único de saída – um “valor de retorno”. As funções em ABL são mais gerais. Estas não exigem parâmetros e também não necessitam retornar um valor.

Elas são invocadas – têm seus códigos executados – usando o nome da função na instrução. Caso a função tenha parâmetros, o nome da função é seguido por parentes com vírgulas separando os parâmetros e fechado com parênteses novamente.

Na prática, as funções servem para executar determinados procedimentos que aparecem diversas vezes no código da aventura. Eles são como mini—*Codes*; podem ter suas variáveis e constantes próprias e seu código próprio. Depois de ter lido a função, a leitura do programa volta para o ponto onde se iniciou a função. Seria mais ou menos assim, de forma figurativa:

#### Código

Caso a aeronave atinja 2500 pés AGL,  
toque o arquivo de som 41;  
mas antes:

- limpe o texto que estava lá;
- zere o som;
- cursor no início;
- apagar fileira;
- texto zero;
- escreva “som 41” na janela de depuração;

Depois disso, continue com...

O que seria uma função aqui? Toda vez que você tiver que limpar o que já estava na espera ou sendo exibido anteriormente, ao invés de digitar as várias linhas de código para isso, crie uma função “*Limpendo texto*” e faça sua chamada sempre que necessitar:

#### Código

Caso a aeronave atinja 2500 pés AGL,  
toque o arquivo de som 41;  
mas antes:

- Função *Limpendo Texto*()
- escreva “som 41” na janela de depuração;

Depois disso, continue com...

As funções vem descritas antes da seção *Code* e podem conter qualquer coisa que você quiser. Não esqueça daqueles lembretes em variáveis sobre o comportamento dessas dentro e fora das funções. Nas aventuras e lições do flight você encontra praticamente tudo o que vai precisar de funções mas o ABL te permite a criação de quantas funções quiser. Se aspecto sempre será parecido com o a seguir:

```
/* Esta rotina exibe uma mensagem de texto
NORMAL por um determinado período de tempo */

function fnSQ_NormalMsg (String strMsg,Number nPause);
code
fnSQ_Private(SQ_MESSAGE,strMsg,nPause,0,FALSE);
endfunction;
```

Aqui você tem uma função que exibirá um determinado texto por um determinado período de tempo na tela do simulador. Isto funciona para o FS2004, aquela telinha azul, que se abre no canto superior esquerdo de determinadas aulas que vem com o FS2004. A instrução começa com **function**, mostrando que é uma função, seguido pelo nome da função: **fnSQ\_NormalMsg**, seguido pelos parâmetros da função entre parênteses: (**String strMsg,Number nPause**): **String strMsg** = temos uma variável de caracteres definida por **strMsg**; **Number** = variável numérica definida em **nPause**. Você pode definir constantes como mensagens tipo:

**Const**  
**msg01** = “Olá colega!”;

Utilizando a função acima em seu *Code*, a coisa poderia ficar assim:

**fnSQ\_NormalMsg (msg01,10)**

Com isso, você faria com que nesse momento fosse executada a função chamada por você para exibir a mensagem 01. Apareceria na sua tela nesse momento algo assim:



As funções em ABL são divididas em três categorias:

**Funções intrínsecas:** Estas funções são parte da implementação do ABL para o Flight Simulator e podem ser usadas em qualquer script do simulador. São implementadas em C e são executados muito rapidamente.

**Funções de objetos unitários:** Funções unitárias são propriedades de objetos que são funções. Também são implementadas em C, mas codificação extra é acrescentada para uso do objeto. Portanto, não são tão rápidas quanto as funções intrínsecas. Apesar disso, ainda

são mais rápidas do que o tipo a seguir.

**Funções declaradas.** Essas funções são definidas no script ABL via seção *Function* dentro de um módulo. Estas funções têm as mesmas capacidades das funções intrínsecas mas são bem mais lentas porque são escritas em ABL, executadas e interpretadas.

Após a seção *Var* no módulo, mas antes de seção *Code*, um módulo pode incluir quantas funções você quiser. Cada seção *Function* declara uma função que pode ser invocada em qualquer parte do script, como o exemplo anterior.

A seção *Function* consiste de uma instrução **FUNCTION**, *Const* opcional e seção *Var*, uma seção *Code* e uma instrução **ENDFUNCTION**. A instrução **FUNCTION** declara o nome da função, parâmetros da função e o tipo de retorno da função.. Se uma função tem parâmetros, eles ficam numa lista entre parênteses, que tem os parâmetros separados por vírgula. Cada parâmetro pode ser usado como qualquer outra variável dentro da seção *Code*, ligado a esse pelo nome. Parâmetros podem ser *arrays* (arranjos). Se uma função tem um tipo de retorno, o tipo é especificado colocando-se dois pontos (:) depois da lista de parâmetros e depois definindo o tipo de retorno. Uma instrução de função é terminada por ponto-e-vírgula.

Função sem parâmetros e sem valor de retorno

**Function MyFunctionA;**

Função sem parâmetros e retorna um número.

**Function MyFunctionB : number;**

Função que aceita um parâmetro e não retorna valor.

**Function MyFunctionC(string s);**

Função que aceita três parâmetros. Retorna o objeto Aircraft.

**Function MyFunctionD(number a, string b, World w) : Aircraft;**

**À função é permitido modificar seus parâmetros**, mas esta restrição geralmente não afeta o valor dos parâmetros fora da função. Quando uma função é chamada, cópias são feitas dos parâmetros usados, da mesma maneira que automaticamente variáveis são alocadas pela seção *Var*. Modificando os valores dos parâmetros dentro da função muda somente o valor da cópia. O valor da cópia é perdido se a execução da função encontra uma instrução **RETURN** ou **ENDFUNCTION**. A única exceção a essa regra é a propriedade dos parâmetros dos objetos. Se um objeto é passado como parâmetro e alguma propriedade é acessada naquele objeto, as propriedades são aquelas do objeto original, não a cópia.

**Funções têm acesso a todas as constantes declaradas no módulo Const e todas variáveis declaradas no módulo Var.**

**Funções têm a permissão de declarar variáveis estáticas na sua seção Var, Essas variáveis mantêm seus valores durante as chamadas.**

#### Exemplo de definição de função

Function AbsoluteIntegerDiff (Number a, Number b) : number;

Var

Number AlsBigger;  
Number difference;

Code

```
AlsBigger = ( a > b );
If (AlsBigger) then
    Difference = a - b;
Else
    Difference = b - a;
Endif;

Return(Difference);
```

Endfunction;

Essa função serve para calcular a diferença absoluta entre dois valores numéricos (*number a* e *number b*). Inicialmente são definidas duas variáveis numéricas: *AlsBigger* e *difference*. *AlsBigger* serve para definir a ordem dos valores na diferença, colocando o maior antes do menor e *difference* o resultado entre eles que será retornado pela função.

## INSTRUÇÕES ESPECIAIS

### Imprimindo

A instrução **PRINT** (*exibir, imprimir, estampar*) pode aceitar uma string ou um number como seu argumento mas não pode misturar os tipos em uma mesma instrução **PRINT**. Entretanto, múltiplas chamadas de função **PRINT** podem ser colocadas na mesma linha. Um método melhor é usar variáveis string temporárias e a instrução **CONCAT** para construir a de saída e depois imprimir essa string. A pasta **[raiz]\Script\Library** inclui funções que você pode usar para criar strings com 2, 3 ou 4 argumentos e para misturar numbers e strings.

Se você quer iniciar a frase de saída numa nova linha, você deve incluir **\n** na instrução **PRINT**. A função da biblioteca *WriteLn* adiciona um caractere de nova linha ao argumento da string e exibe a string.

Todos as instruções dos procedimentos seguintes produzem o mesmo resultado (se a biblioteca for incluída no script):

Print ("A Altitude atual é de ");

print (nAltitude);

Concat (strTemp, "\n");

Print (strTemp);

Números têm o formato: ###.#####e###.

#### Interface de depuração

O comando embutido **PRINT** é primariamente útil para depurar scripts, já que o parâmetro é exibido numa janela de console, fora das janelas de visão do Flight Simulator.

Habilite a apresentação da depuração usando a diretiva **#PRINT\_ON** antes de qualquer instrução **PRINT** e desabilite-a usando a diretiva **#PRINT\_OFF**. Sintaticamente, a instrução **PRINT** é uma função que aceita um único argumento number ou string.

### Bibliotecas

ABL suporta o conceito de códigos reutilizáveis. Frequentemente sub-rotinas usadas podem ser colocadas numa biblioteca. Essas funções podem ser incluídas em outros programas. Tome cuidado quando desenhando e usando funções importadas para se assegurar que variáveis globais foram definidas e que a ordem e forma foram observadas.

Quando uma diretiva **#INCLUDE** é encontrada, a função da biblioteca é lida exatamente como se essa fosse parte do script principal. Quando instalamos o FS2004 várias bibliotecas são instaladas na pasta **[raiz]\Script\Library**. Vamos discutir abaixo um pouco das bibliotecas mais usadas e mais importantes e como elas podem ser úteis aos seus projetos, caso queira incluí-las. Vale a pena dar uma estudada nelas para aprendizado. Essas funções (rotinas) incluídas com a instalação do FS2004, são obrigatórias em sua maioria para o funcionamento das lições e aventuras que vem com o simulador. A Microsoft preferiu fazer dessa maneira para provavelmente economizar tempo e agilizar a criação das aventuras, já que ela utiliza o mesmo padrão para todas. Você não precisa utilizá-las caso não queira; você pode escrever todo o código em um arquivo \*.abl somente, onde estará seu *code*, *var* e *const*. As aventuras criadas pela *FSAdventures* e pela *Perfect Flight* não utilizam as bibliotecas do FS2004.

#### Bibliotecas Nativas do FS2004

##### 1) Math.abl

Contém funções "matemáticas", como o próprio autor descreve, com fórmulas para conversão de metros em pés e vice-versa, calcula da diferença entre dois valores entre outras.

##### 2) Messages.abl

Contém funções (rotinas) para exibição de mensagens de texto

##### 3) SimQueue.abl

Uma das bibliotecas mais importantes e utilizadas na simulação. Contém rotinas de exibição de texto, texto NORMAL, DE AVISO e DE ALERTA, com e sem som, enfileiramento de sons, mudanças de Estado, etc. Cuidado com as rotinas privadas (*private*); não as utilize diretamente nos códigos.

##### 4) stdCases.abl

Casos (*cases*) básicos que são incluídos em todas as lições e aventuras do FS2004, como retardar em 10 segundos o início da aventura, até a simulador já ter carregado o voo.



### 5) *stdChkTolerances.abl*

Rotinas complexas que servem para avaliar níveis de tolerâncias que possam ser incluídos nas lições e aventuras, tipo altitude máxima, velocidade no ar máxima, o que fazer se ultrapassar limites, etc. Muitas aventuras não incluem tipo algum de tolerância, ao contrário dos vãos de certificação que são “pura tolerância”.

### 6) *stdhousekeeping.abl*

Contém inicialização de variáveis comuns importantes para o funcionamento adequado de todas as aventuras. Importante para todos os casos.

### 7) *stdinit.abl*

Contém inicialização de variáveis **globais** importantes para o funcionamento adequado de todas as aventuras. Importante para todos os casos.

### 8) *stdkonstants.abl*

Desnecessária para seus vãos. Você terá de definir suas próprias constantes mas vale a pena abrir e estudar pra entender. Você verá uma lista enorme de casos aqui — CaseCal1=1010; CaseCal2= 1011; etc. Essas constantes podem ser definidas na seção **Const** de seu script (arquivo \*.abl) da aventura principal. Como a Microsoft criou um modelo de funcionamento das lições e aventuras baseado em SWITCH/CASE, os casos (cases) devem ser definidos na seção const ou num arquivo separado como esse com as constantes, e ser chamado via #INCLUDE do script principal. O valor após o sinal de equivalência (=) é aleatório, vale colocar numa ordem útil para não confundir e comece com um valor alto (1000, 10000, ...) para não coincidir com outros valores de outras variáveis ou constantes: CaseAgurada=1000; CaseDecola=1001; etc.

### 9) *stdvars.abl*

Variáveis globais usadas em todos os scripts e também em diversas bibliotecas. Importante ser lida para poder entender a definição de diversas variáveis encontradas em outros elementos do script e nas outras bibliotecas.

### 9) *stringfns.abl*

Arquivo pequeno com funções de string (caracteres) utilizadas em todos os scripts, como por exemplo, *Writeln*, a mais importante dessa biblioteca.

### 10) *throttle.abl*

Diversas rotinas criadas para trabalhar a aceleração da aeronave, como por exemplo, ajustar a aceleração a uma rotação constante, transferir o controle da aceleração para manual, desligar os motores, etc.

### 10) *fuserevents.abl*

Define as variáveis globais para utilização do teclado com as combinações <CTRL + > ou <CTRL + >. Ainda não sei se nesta versão do FS é permitido a utilização de outras teclas, como era quando programávamos com APL.

Além dessas bibliotecas, poderemos incluir arquivos de mensagens (\*.msg) que nada mais são do que arquivos com uma coleção de constantes que podem ser incluídas na seção *Cosnt* de seu script. Esses arquivos ficam nas pastas e sub-pastas a partir de *[raiz]\Messages*. Em *[raiz]\Messages\lib* você vai encontrar um monte de mensagens de tolerâncias utilizadas nas aventuras do FS. Se você quiser, em todos os arquivos de mensagens ou nas constantes de mensagens que você quiser adicionar no seu script, tudo o que estiver entre as aspas ( " " ) pode ser reescrito ou traduzido:

*sDing1="Você não acendeu as luzes de pouso antes da decolagem.";*

## Considerações de Performance

Os scripts devem minimizar seu impacto na performance da simulação utilizando interfaces o mínimo possível. Há sempre uma sobrecarga em cada chamada de entrada ou saída do script. Você pode melhorar performance minimizando o número de chamadas para frente ou para trás. Por exemplo, se um script executa várias operações profundas em um modelo de objeto, é mais rápido criar uma variável para armazenar o objeto do que recuperar o objeto cada vez que for necessário. Você entenderá mais profundamente o conceito e funcionamento dos objetos em seção mais adiante, mas já dá pra ter idéia de como as coisas funcionam. Aqui está um exemplo de um jeito lento e rápido de se executar a mesma coisa:

#### Code

```
World.PlayerAircraft.Autopilot.HeadingHold = TRUE;
World.PlayerAircraft.Autopilot.Heading = 180;
World.PlayerAircraft.Autopilot.Master = TRUE;
```

Isto é lento porque o mecanismo do script precisa resgatar o objeto *Autopilot* toda vez. Se o script for estruturado da maneira seguinte, seguindo sempre a hierarquia na organização dos objetos, a melhora na performance será bem

#### Var

```
AutoPilot AP;
aircraft acPlayer;
world w;
```

#### Code

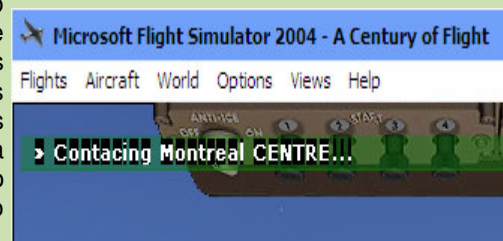
```
w=GetWorld; // World
acPlayer=w.GetAircraftByID(0) // World.PlayerAircraft
AP=acPlayer.Autopilot; //Faz a variável segurar o objeto Autopilot
AP.HeadingHold=TRUE; //World.PlayerAircraft.Autopilot.HeadingHold
AP.Heading=180; //World.PlayerAircraft.Autopilot.Heading
AP.Master=TRUE; //World.PlayerAircraft.Autopilot.Master
```

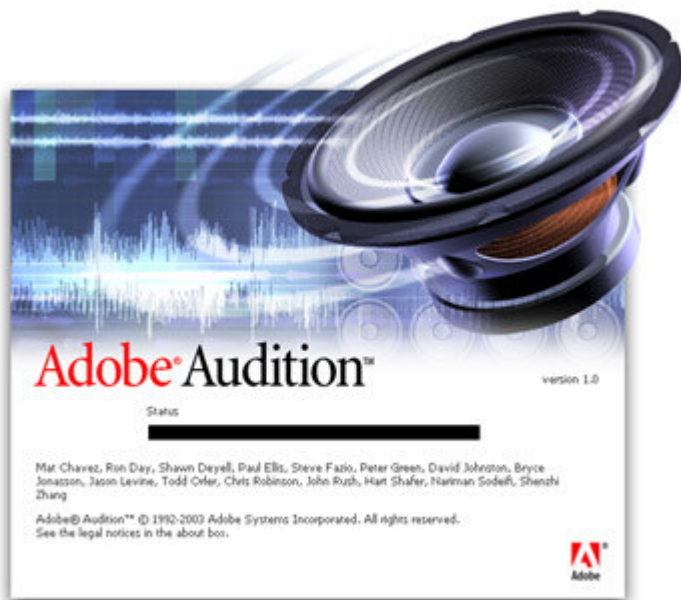
clara.

Também seria uma boa idéia retardar o acesso às propriedades do sistema de vôo da aeronave por cinco ou dez segundos depois do vôo iniciar. Embora um script inicie com a primeira taxa de quadros renderizada, o mecanismo de simulação requer 5 a 10 segundos para carregar e estabilizar. Essa rotina você encontra pronta na biblioteca *stdCases.abl*. Qualquer propriedade da aeronave que você acesse durante o período de estabilização pode ser somente parcialmente usado.

## Arquivos de Som e Legendas

Arquivos de som são tocados nas aventuras e lições do Flight Simulator via método *PlaySound* do objeto *Shell*. O formato recomendado é 22,050 samples per second (amostragem por segundo), PCM adaptative (ADPCM.). Caso arquivos de som de diferentes formatos ou taxas de amostragem forem tocados conjuntamente, a qualidade do som final vai ficar comprometida. Uma exceção pode ser feita para arquivos de comprimento — zero (zero-length files) que estão presentes somente para as legendas. Essas legendas, como a da figura ao lado, estão inseridas num arquivo \*.wav de som.

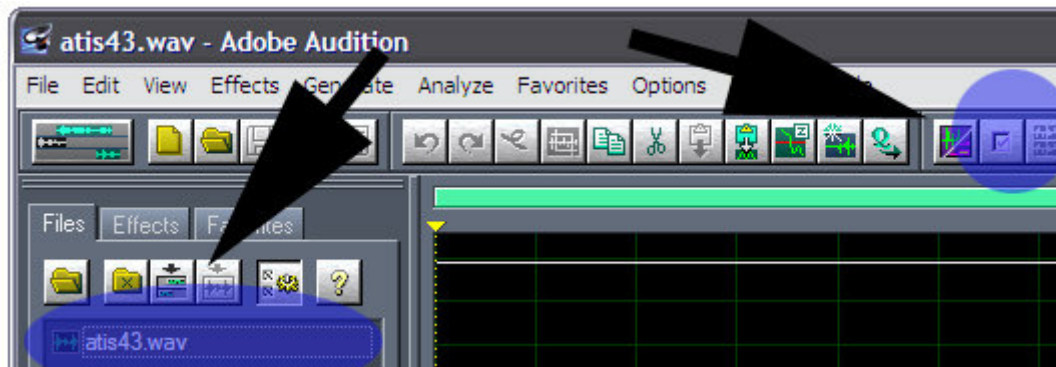




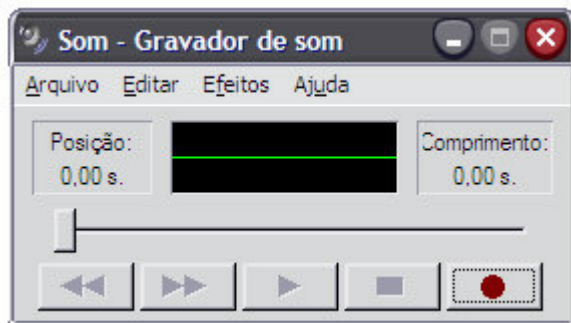
O Adobe Audition é o mesmo conhecido editor de arquivos musicais chamado **CoolEdit**, que foi comprado pela Adobe mas não mudou nada no seu funcionamento. Vou colocar aqui como você pode gravar as mensagens de texto nos arquivos de som. Há duas possibilidades de isso ocorrer: 1. usar um arquivo de comprimento zero, apenas para exibir mensagem de texto sem som algum; 2. usar um arquivo com texto e som conjuntamente. Para gravar o som, você pode utilizar os programas acima citados — não vou entrar em detalhes desses processos — e depois escrever o texto. Após aberto ou gravado o arquivo de som, você terá uma lista à sua direita com o nome do(s) arquivo(s) a ser(em) editado(s). O primeiro da lista será aquele que estará ativo ao trabalho.

Abaixo da barra de menus, nos botões de comando, você encontrará um ícone com um pequeno quadrado no seu interior com uma

Essas legendas ficam na porção **INAM** do arquivo wave, e este será mostrado no topo da tela à medida que o arquivo wave é tocado. Se o indicador **Persist** é acionado quando o arquivo é tocado, o texto permanecerá na tela até que este seja limpo pela entrada de outro texto ou por tocar um arquivo wave que não tenha legendas. Caso contrário, o texto permanece na tela por aproximadamente 15 segundos. Você pode inclusive gravar o som para que esse seja exibido juntamente com o texto. Ainda não descobri um



meio, nem consigo informações nos fóruns, de como fazer a mensagem rolar na tela, com o quando você recebe as informações do ATIS. Se

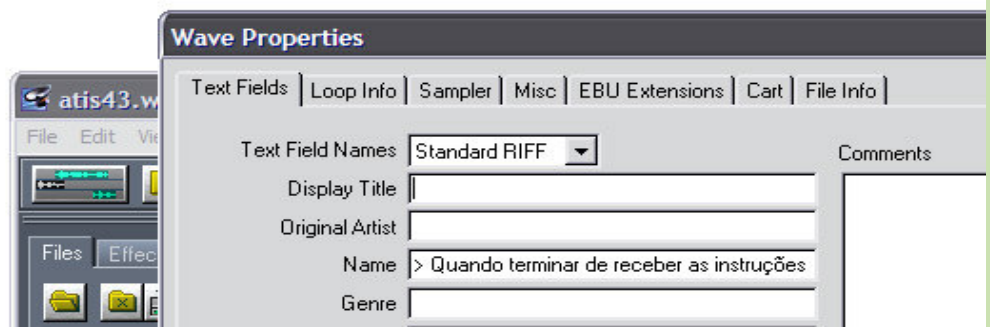


a mensagem for muito grande, ela será quebrada em mais de uma linha.

Para gravar os arquivos de som, você pode utilizar qualquer software de gravação e edição de som, inclusive o **gravador de som** que vem nos acessórios do Windows. É uma ferramenta básica, com poucos recursos, mas que na falta de uma ferramenta mais completa, pode ser útil. O único problema é que com este gravador você não conseguirá editar as frases de texto nas porções INAM dos arquivos wave. Para isso, você deve utilizar um programa mais "profissional", algo do tipo **Adobe Audition 1.0**, que é excelente e também pago e caro.

Após aberto essa janela, você vai digitar o texto que quer que apareça junto ao arquivo de som, ou somente o texto, em **Name**. Os outros campos nada incluem e nada modificam na apresentação da legenda, nem na execução do som. Depois de editado o texto, é só sair e salvar seu arquivo.

Para criar um arquivo somente pra exibir legendas, clique em **File > New >** e em **New Waveform** escolha Sample Rate **6000**, Channel **Mono** e Resolution **8 bit**.



Esse é um formato mínimo, que aceita muito bem as legendas e tem um tamanho pequeno, leve para ser carregado pelo simulador. Para som, utilize o formato já recomendado de **22050**, **Mono** e **8 bit**.



## Tocando um Arquivo de Som

Se o método do ABL *Shell.PlaySound* não puder encontrar o nome do arquivo especificado, no primeiro parâmetro, nenhuma mensagem de erro será gerada. Simplesmente o som não será tocado. Por colocar o método *PlaySound* numa função local que irá exibir o nome do arquivo na janela de depuração, o programador pode simplesmente esperar por um nome de arquivo que é seguido por silêncio ao invés do som esperado.

O método *PlaySound* requer uma sintaxe do nome do arquivo no estilo da linguagem C, isto é, os caminhos devem ser especificados com duas barras para esquerda. A linha em ABL que especifica `c:\directory\subdirectory\wavefile.wav` é `c:\\directory\\subdirectory\\wavefile.wav`. Omitir a segunda barra \ é o erro mais comum em se especificar nomes e caminhos.

O método *PlaySound* procura pelo nome do arquivo na raiz do diretório onde o Flight Simulator foi instalado. Se o caminho não incluir \\ o interpretador APL adicionará `C:\\Program Files\\Microsoft Games\\Flight Simulator 9\\` à sequência do argumento.

```
// Para executar um arquivo de som com rotinas padrão do FS2004
// Abaixo a rotina que está em SimQueue.abl

function fnSQ_Play(String Filename);
var
    String strFileSpec;
code
    strFileSpec = fn3StrBuild(strWavBase,Filename, ".WAV");
    fnSQ_Private(SQ_Play, strFileSpec, 0, 0, FALSE);
endfunction;

// No script, insira a instrução com o nome do arquivo sem extensão
fnSQ_Play("som01");

// Para interromper o arquivo de som, toque um arquivo nulo.
Shell.PlaySound ("", FALSE, 0);
```

## Mais Legendas 1

Além das legendas associadas aos arquivos de som, discutidos até agora, a partir do FS2004 temos uma nova forma de exibir mensagens de texto. São janelas parecidas com as do ATC e permitem exibição de até cinco linhas com mais de 50 caracteres cada, como a imagem abaixo.



Essas mensagens podem ser de três tipos, conforme funções já previamente definidas pelo FS2004: NORMAL (escritas em branco), WARNING (amarelo) e ERROR (em vermelho). Também são reprogramáveis. Elas podem ser chamadas no script da seguinte maneira:

**fnSQ\_NormalMsg(msg01,5);**

Aqui você chama uma mensagem NORMAL que aparecerá em branco e permanecerá por 5 segundos. A constante **msg01** deverá ser definida na seção *Const* de seu script ou de outro script \*.abl ou \*.msg, desde que seja chamada no início com **#INCLUDE**.

**Por exemplo:**

```
Const
msg01 = "Iniciando aventura";
Code
fnSQ_NormalMsg(msg01,5);
```

É importante lembrar que estamos considerando que neste caso você está usando as definições de variáveis e funções que já vem com o FS2004 e foram incluídas no script principal ou chamadas por **#INCLUDE**.

**fnSQ\_WarningMsg(msg01,5);**

Essas mensagens serão exibidas em amarelo.

**fnSQ\_ErrorMsg(msg01,5);**

Essas mensagens serão exibidas em vermelho.

Você pode criar suas próprias rotinas para exibir mensagens de texto e/ou limpá-las. Abaixo, um exemplo com duas funções de Fermin Fernandez para chamadas de texto prontas para usar, onde o texto é adicionado diretamente na chamada da função. As quebras de linha são definidas pela inserção de **"\n"** nos pontos de quebra.

Após a parte teórica deste guia, teremos alguns exemplos de criação de aventuras e esses detalhes de chamadas de arquivos de som com ou sem legendas serão lembrados. Com a apresentação de legendas deste segundo modo, você pode também gravar um arquivo de som com o

```
// Escreve uma mensagem para uma janela no FS2004 e depois
// a limpa. Autor: Fermin Fernandez
```

```
Function fs2004textmessage (string fs2004textmessage, number seconds);
var
code
    s.TextMessage(fs2004textmessage,1);
    msgDisplayTime = seconds;
EndFunction;
```

```
Function fnClearTxtMsg;
s.TextMessage(fs2004textmessage,1);
msgDisplayTime = seconds;
EndFunction;
```

```
Para usar no script, faça como no exemplo abaixo:
fs2004textmessage ("Escreva na linha um \nNa linha 2 \nNa linha três",15);

Para limpar mensagens prévias de texto:
fnClearTxtMsg;
```

conteúdo do texto e fazer com que este seja executado com o texto, necessitando de duas chamadas: uma para o texto e outra para o som. No primeiro método de apresentação de legendas, os dois vinham incorporados no mesmo arquivo, não necessitando o texto possuir o mesmo conteúdo que o som.

## Mais Legendas 2

Outra forma de exibir mensagens de texto no FS2004 é utilizar o método *TextMessage* do objeto *Shell*, assim::

**TextMessage(string strText, int iPriority)**

Aqui a string **strText** também pode ser uma constante como na forma anterior e tem adicional uma informação de prioridade **iPriority** que define ao cor da informação a ser exibida:

```
0 – Normal. Padrão em branco
1 – Alerta. Padrão em amarelo
2 – Importante. Padrão em vermelho.
```

**Shell.TextMessage(Mensagem01,2)**

Para apagar a mensagem atual, use uma instrução vazia:  
**Shell.TextMessage("",0)**



## Frequências de Rádio

As frequências de rádio retornam valores dos objetos de rádio como números representando a frequência real em megahertz (para VHF) ou kilohertz (para ADF). Devido ao fato de que a maneira como os valores dos pontos flutuantes são armazenados, essas frequências raramente são as mesmas que você vai ajustar no rádio. Este é o caso onde um teste para `nFreq == 115.80` não é recomendado. Ao contrário, um teste para `Diff (nFreq, 115.80) < 0.001` é mais confiável. Lembrando que entre parênteses, a equivalência é dupla `"=="`, e fora, simples `"!="`.

```
if (nIASNow == 200) then;
    nIASNow = 200;
```

## Trabalhando com Direção e Rumos

A direção da aeronave retorna uma direção real em graus. Rumos são computados pelo método *BearingTo* do objeto *Position* (posição) e pelos métodos *VORRADIO* e *ADFRADIO*. Esses métodos retornam o rumo real da aeronave para o objeto/estação em graus. Para converter esses valores para direções e rumos magnéticos, subtraia a variação magnética da aeronave. *Magnetic variation* é um dos atributos do objeto *Position*. Para fazer o piloto automático guiar para a localização/estação, obtenha o rumo real, converta para magnético e depois defina *AutopilotHeading* com esse valor.

Lembre-se de que à medida que a aeronave se aproxima da localização, o rumo começa a mudar muito rapidamente. Determine empiricamente uma área radial que você irá considerar sobre a estação, levando em conta a altitude da aeronave e a velocidade de solo. Use o método *DistanceTo* para determinar quando a aeronave cruzou o raio dessa esta área e mova-se para a próxima estação ou latitude/longitude. Este procedimento previne a aeronave de circular a estação sem realmente atingi-la.

## Nomes de Funções Reservadas

Cinco nomes de funções especiais são reservados para uso com o Flight Simulator 2002. Estas são: *Init*, *On\_Stall*, *On\_Crash*, *On\_Exit*, *OnUserEvent1* e *OnUserEvent2*. Essas funções são chamadas assincronicamente pelo Flight Simulator quando o evento associado ocorrer. As sub-rotinas chamadas por esses eventos devem ser curtas e retornar o controle ao loop principal do simulador o mais rápido possível. O método preferido para manipular esses eventos é definir um sinal nas variáveis globais e depois retornar. Esse sinal pode ser checado quando o loop do script inicial começar a ser executado. Como qualquer outra função, essas funções devem ter uma declaração **FUNCTION**, uma instrução **CODE** e uma instrução **ENDFUNCTION**. Essas funções não podem retornar um valor para o Flight Simulator e essas não devem ter um tipo de retorno declarado. Note que *OnStall* é chamado antes que a sirene ou aviso de estol seja acionado na simulação. O ponto de entrada *OnCrash* não é chamado se não for checado "ignore crash" em ajustes de realismo no simulador.

## Depurando

A depuração primária é acompanhada pela lição ou aventura de vôo em que você está trabalhando e observando os resultados. Esse processo pode ser entediante especialmente quando empreendido em achar uma falha num vôo longo. Você pode achar útil inserir uma instrução **PLAYSOUND** no início de uma sequência que você quer depurar. Usar um único som prende sua atenção, por exemplo, um arquivo que contenha a frase "Ok, ouça isso !". O recurso de exibição **DEBUG** também é bastante útil quando depurando um script.

Depurar o script só é possível se a diretiva **#DEBUG\_ON** estiver presente no script. Uma janela DOS é criada (se esta já não existir) e as informações de depuração são exibidas lá. A primeira vez em que a janela de depuração é criada, o Flight Simulator entra em pausa (se o simulador estiver configurado em Settings/General em "Pause on Task Switch". NT: Geralmente eu configuro meu FS para operar com essa função desabilitada.).

Quando o script é carregado, o interpretador rastreia o script por erros na sintaxe e problemas como variáveis não declaradas, funções perdidas, etc. Se alguns desses erros forem detectados, ou se um erro de execução ocorrer, informações sobre o(s) erro(s) será(ão) mostrada(s) na janela de depuração. Erros de linguagem (i.e., não lógico) fazem o script interromper a execução. É importante notar que o vôo, entretanto, não pausa ou termina.

A diretiva **#PRINT\_ON** envia instruções lógicas de depuração para a janela de depuração (via instrução **PRINT**). Para limitar o resultado de depuração de um seguimento do script, você pode controlar o resultado com a diretiva **#PRINT\_OFF**.

Sem ter uma diretiva habilitada, erros de sintaxe vão evitar que o script seja carregado e o único retorno para o usuário será uma caixa de mensagem informando ao usuário que a "the adventure is corrupted." Escolhendo OK faz com que você continue o vôo, mas o script não será executado.

## Notação Húngara e Nomes de Variáveis

Em termos simples, Notação Húngara é uma técnica para fazer o software ser mais fácil de ser escrito, mantido e depurado. É um método padronizado de nomeação de variáveis que permite a qualquer um ler seu software para determinar de imediato que tipos de variáveis serão usadas e outros certos aspectos do código. Funciona mais ou menos assim: *O primeiro caractere de uma variável é o tipo da variável, em letra minúscula. O resto do nome das variáveis é capitalizado, juntando texto descrevendo a variável.*

No caso do ABL, uma convenção útil é **n** para números, **b** para Booleanos, **ac** para Aircraft, **str** para string, etc. O restante do nome da variável é a descrição do propósito da variável. Alguma esquisitice é permitida, desde que o principal se mantenha claro. A variável que representa o cockpit, que é meu escritório, é do tipo *cp* (cockpit) e nome *Office*. O nome inteiro é, entretanto, *cpOffice*.

Fidelidade total ao sistema pode fazer com que algumas instruções do programa pareçam enigmáticas de imediato; mas que são na verdade fácil de decodificar. Por exemplo:

```
For nLooper = 0 to nEngineCountm1 do
    ecEngines[nLooper] = cpOffice.EngineController(nLooper);
    eMotors[nLooper] = acMe.Engine(nLooper);
endfor;
```

Um loop está sendo executado. O contador é um número chamado *nLooper*. Seu valor inicial é 0 e deve ser executado até que seja igual ao número de representações *EngineCount-1*.

Um arranjo do objeto controlador do motor está sendo inicializado para o objeto do cockpit *Engine Controller* que corresponde ao número do loop.

Um outro arranjo de objetos de motores está sendo inicializado para o objeto aircraft, novamente correspondendo ao número do loop.

# O Modelo de Objeto do Flight Simulator

Esta parte do documento descreve o Modelo de Objeto (OM) para a versão ABL do Flight Simulator. Começamos com uma discussão geral do OM. Depois, uma lista detalhada de **objetos**, **propriedades** e **métodos** usados no Flight Simulator.

## Objetos do Flight Simulator

Embora ABL não seja uma linguagem de programação verdadeiramente orientada a objeto, certos aspectos no Flight Simulator podem ser manipulados como **objeto**. Por exemplo, **o objeto Shell é a interface ABL para o jogo Flight Simulator**; a maioria das aeronaves têm um ou dois objetos **VORRADIO**, os quais fornecem um caminho para a interface ABL com os rádios de navegação VHF.

Alguns objetos têm **propriedades**, alguns têm **métodos** e alguns ambos. Um exemplo de propriedade é *Aircraft.Autopilot*, onde *Autopilot* é a propriedade de *Aircraft* que retorna um sub-objeto ao piloto automático. Já um exemplo de método é *Shell.PlaySound()*, que vimos nas discussões sobre execução de arquivos de som: *PlaySound* é um método do objeto *Shell*.

Alguns objetos devem ser posicionados a cada passo através do script em ABL. Por exemplo, objetos **VORRADIO** não são estáticos, não possuem sempre o mesmo valor, e devem ser encontrados cada vez que forem usados.

Objetos que podem ser posicionados uma única vez e não vão mudar durante a execução devem ser atribuídos a variáveis estáticas durante a inicialização. Por exemplo, o objeto *Autopilot* é estático e deve ser posicionado somente uma vez durante a inicialização. Uma variável do tipo *Aircraft* é declarada e uma outra do tipo *Autopilot*. Durante a inicialização, a variável *Aircraft* é igualada à aeronave do usuário e *Autopilot* é igualada ao piloto automático da aeronave. Por exemplo:

```
Var
World    wTheWorld;    //Este planeta
Aircraft acMe;         //Minha aeronave
Autopilot AP;          //Meu piloto automático
```

```
Function Init;
Code
wTheWorld = GetWorld();    // Em qual planeta estamos agora?
acMe = wTheWorld.AircraftByID(0); // A aeronave do usuário
AP = acMe.AP;              // Meu piloto automático
EndFunction;
```

O grampo de marcação do curso (*heading bug*) pode agora ser modificado alterando a propriedade *AP.Heading*. Isso requer muito menos tempo para execução do que o código alternativo:

```
Ap.Heading = 180;          ao invés de:
GetWorld().AircraftByID(0).AP.Heading = 180;
```

Na programação em ABL, para você acessar os modelos de objetos, você vai precisar seguir uma hierarquia, partindo sempre do topo. No topo da hierarquia dos objetos do flight Simulator temos dois objetos principais: **Shell** e **World**. Eles fornecem acesso a diferentes aspectos da simulação. Por exemplo, a seguir veremos como recuperar o objeto *Aircraft* da aeronave do usuário.:

```
Var
Aircraft acPlayer;    // declare a variável da aeronave
world w;              // declare a variável do mundo
```

```
Code
w = getWorld;          // recupera o mundo virtual
acPlayer = w.AircraftByID(0) // recupera a aeronave
```

## Visão Geral do Modelo de Objeto

O Modelo de Objeto (OM) é mostrado pelo Flight Simulator através dos scripts em ABL das seguintes maneiras:

**Extensões Intrínsecas** — Funções globalmente disponíveis que têm sido adicionadas à linguagem ABL. Estas extensões são definidas na documentação da linguagem ABL.

**Biblioteca de Objetos** — Objetos que expõem um conjunto de funções e propriedades que representam elementos da simulação. Esses objetos são explicitamente registrados pelo script principal quando o mecanismo do script é iniciado.

Note que o modelo de objeto atual tem seu foco em fornecer a funcionalidade que a maioria dos scripts requerem sem expor todas as variáveis da simulação. ABL não é uma linguagem realmente orientada a objeto e, ao invés disso, confia numa coleção de bibliotecas que determinam um conjunto de funções associadas. Em uma sintaxe ABL comum temos a maneira como o método *Fuel* no objeto *Aircraft* é chamado. O *idx* é o índice para o elemento combustível (*fuel*) que você está interessado e *aircraftHandle* é uma manobra para o objeto *Aircraft* em particular ser consultado.

### Aircraft.Fuel(aircraftHandle, idx)

O Flight Simulator tem estendido o mecanismo de nome-espaco das bibliotecas em ABL para criar uma sintaxe orientada-por-objeto para que o autor do objeto possa escrever:

```
Var
aircraft acMyPlane;
fuel fuelMyTank;
world w;
number nFull;
```

```
Code
w = getWorld;
acMyPlane = w.AircraftByID(0);
fuelMyTank = acMyPlane.Fuel(0);
nFull = fuelMyTank.PercentFull;
```

onde a especificação do tipo *acMyPlane* como um objeto (definido na biblioteca principal) permite que a função *Fuel* se refira à função *Aircraft Library* de maneira correta e *aircraftID* de *acMyPlane* seja preenchido “por trás dos bastidores”.

Note que devido a limitações no analisador, uma instrução como: *acMyPlane.Fuel(0).percentFull* onde uma propriedade é tratada com método, não vai funcionar como uma simples instrução. Ao invés disso, as linhas devem ser quebradas com no exemplo anterior. Esta convenção sempre permite que a especificação do OM a seguir seja apresentada na forma de uma hierarquia de objetos que resume várias dimensões da simulação e diz que você está adequado a interagir com ele. No topo do nível de hierarquia, Flight Simulator mostra apenas dois objetos:

- O objeto **World** contém informações sobre tudo o que acontece no “mundo da simulação”; incluindo acesso à aeronave do usuário, condições meteorológicas e informações sobre um voo ou aventura em particular em execução.
- O objeto **Shell** contém informações sobre a versão do Flight Simulator e mostra um número de métodos que criam dados de objetos e registradores de tempo.

## Referência do Modelo de Objeto

Esta seção contém as referências de propriedade e objeto para cada um dos objetos mostrados no Flight Simulator.

### Objetos de Posição

Latitude e longitude são representados como pontos-flutuantes em graus decimais ao invés de graus-minutos-segundos; por exemplo, 48°N 30' 00" é 48.50.

| Nome da Propriedade   | Tipo da Propriedade | Unidade       | Atributos                                                                                                            | Descrição                                                                                                                                                                                                                                                                                                                               |
|-----------------------|---------------------|---------------|----------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Latitude<br>Longitude | double              | Graus         | Recupera<br><br>(Significa que você pode obter e trabalhar com esses valores, mas não pode ajustá-los ou defini-los) | Latitude e Longitude são representados em graus decimais. E115° 23' 54.29" é representado como 115.398413885556.<br><br>A fórmula é $DDD+(MM/60)+(SS.SSS/3600)$ . Longitude Leste e Latitude Norte são positivas. Longitude Oeste e Latitude Sul são negativas. Isto é compatível com práticas aceitáveis de navegação náutica e aérea. |
| Altitude              | double              | feet<br>(pés) | Recupera                                                                                                             | A altitude é armazenada como pressão acima do nível do mar (MSL) ou altitude em QFE. Esta é a altitude mostrada no altímetro se 29.92 em Hg for ajustado na janela de Kollsman. A pressão barométrica deve ser corrigida (100 pés por 0.1" Hg) para se obter a altitude indicada.                                                       |
| MagneticVariation     | double              | graus         | Recupera                                                                                                             | Graus de variação magnética nesta latitude, longitude e altitude. A diferença em graus entre o norte magnético atual e o norte real. Subtraindo esse valor do rumo/curso real temos o rumo/curso magnético                                                                                                                              |

| Nome do Método                                  | Tipo de retorno | Unidade | Descrição                                                                                                                                                            |
|-------------------------------------------------|-----------------|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DistanceTo (number Latitude, number Longitude ) | double          | Metros  | Retorna a distância em metros à posição especificada. Retorna a distância em linha reta no solo (não circular) ao ponto especificado por <i>Lat</i> and <i>Lon</i> . |
| BearingTo(number Latitude, number Longitude )   | double          | Graus   | Retorna o rumo real em graus à posição especificada. Retorna o rumo real em linha reta (não circular) ao ponto especificado por <i>Lat</i> and <i>Lon</i> .          |

Por definição, os objetos *LatLonAlt* representam posições absolutas no mundo e estão sempre na taxa de quadros de referências do mundo.

Os objetos *LatLonAlt* recebem valores quando são criados e esses dados não mudam a não ser que explicitamente operados pelo script. Por exemplo, *Aircraft.LatLonAlt* retorna um objeto *LatLonAlt* contendo a posição daquela aeronave naquele momento específico. Se você quiser a posição daquela aeronave no futuro, você deve chamar *Aircraft.LatLonAlt* novamente para obter um novo objeto *LatLonAlt* com um novo valor. (A primeira variável não atualiza dinamicamente; não tem ligação ao objeto *Aircraft* de origem). Isto significa realmente que todos os objetos *LatLonAlt* podem ser reutilizados. Se você tem uma variável *LatLonAlt* de um cálculo prévio, estará OK definir novos valores e reusar o objeto. Você poderá sempre criar um novo objeto via *System.NewLatLonAlt* sempre que precisar.

### Os Objetos de Simulação

Os objetos de simulação são os objetos que realmente expõem o modelo de objeto para os itens globais (*world items*) na simulação. Por exemplo, esses

objetos incluem aircraft, autopilots, engines e radios.

Note que a lista a seguir de objetos de simulação não pode ser declarada como variáveis estáticas. Para acessar esses objetos numa função, você deve consultar a aeronave para eles cada vez que a função for executada.

- Position
- Cockpit
- AIController
- LandingGear
- AutoPilot
- EngineController
- Engine
- Fuel
- ADFRadio
- COMRadio
- VORRadio
- TransponderRadio



## O Objeto Shell

| Nome da Propriedade | Tipo da Propriedade | Unidade | Atributos | Descrição                                                                                                                                                                                                                                          |
|---------------------|---------------------|---------|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| InputDevice         | Int                 | --      | Recupera  | Retorna um código indicando o dispositivo de entrada.<br>0 – somente teclado<br>1 - Joystick habilitado<br>2 – Joystick com ForceFeedback habilitado                                                                                               |
| ProductVersion      | Int                 | --      | Recupera  | Retorna um inteiro com o número da versão da aplicação principal. Isto é do formato [versão maior * 100 + versão menor].                                                                                                                           |
| ScriptingVersion    | Int                 | --      | Recupera  | Retorna a versão do modelo de objeto do script. Note que isso pode ser atualizado mais freqüentemente do que os produtos. Isto é do formato [versão maior * 100 + versão menor]. Por exemplo, um modelo de objeto que tem a versão 1.0 retorna 100 |

| Método                                                                        | Tipo de Retorno | Unid     | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|-------------------------------------------------------------------------------|-----------------|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| enableSlewMode(BOOL Enable)<br>string Message)                                | --              | --       | Habilita SlewMode se bEnable for verdadeiro. Se bEnable                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| AdvanceSoundQueue                                                             | --              | --       | Avança a fila de som para o próximo arquivo wave. Se a fila                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| GetRemainingSoundTime                                                         | Int             | Segundos | Retorna a quantidade de tempo faltando da fila de som do início da fila ao final da execução. Se a fila estiver vazia,                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| TextMessage(string strText,--<br>int iPriority)                               | --              | --       | Mostra uma mensagem de texto com não mais do que duas linhas numa janela transparente na tela principal. Este é um texto usado para fornecer informação duradoura. O texto ficará na tela até ser limpo. Isto pode ser feito usando essa função chamando uma mensagem vazia:<br><b>shell.TextMessage("",0);</b> o parâmetro priority é usado para definir a cor do texto:<br>0 – Normal. Padrão em branco                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| PlaySound(string SoundFilename,<br>BOOL fPersistMessage,<br>int VolumeLevel ) | --              | --       | Toca o arquivo *.wav específico. Se um som já estiver sendo tocado quando <i>PlaySound</i> for chamado, então o som específico é adicionado à fila de sons e será tocado numa próxima oportunidade. Se TRUE for especificado em <i>fPersistMessage</i> , qualquer mensagem atualmente na fila será apagada e esta mensagem será exibida e permanecerá visível até ser limpa. Esta pode ser limpa chamando novamente uma função <i>PlaySound()</i> , mas sem som. Somente uma mensagem persistente pode ser tocada por vez e enquanto é exibida, nenhuma legenda subsequente será vista. Este aviso é preferido de ser usado para mostrar referências ou instruções de texto que irão durar até um evento ocorrer. Por exemplo, para exibir a mensagem "Vire para a direção 355" e manter a mensagem na tela até que a direção seja atingida. VolumeLevel deveria ter passado um valor de 0, que usa o valor padrão de som. Níveis diferentes de zero não são suportados pelo Flight Simulator. |

## O Objeto World

O objeto *World* reúne informações sobre o mundo dentro da simulação.

| Nome da Propriedade | Tipo da Propriedade | Unidade | Atributos           | Descrição                                                                                                                                                                                                                                                                                              |
|---------------------|---------------------|---------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pause               | Boolean             | --      | Recupera/<br>Ajusta | Permite ao script pausar a simulação. Retorna TRUE se a simulação está pausada, FALSE se está sendo executada. Definindo em TRUE, pausa a simulação e não terá efeito se a simulação já estiver pausada; definindo em FALSE, retorna à simulação e não terá efeito se esta já estiver sendo executada. |

| Método                | Tipo de Retorno | Unidade | Descrição                                                                                                                                        |
|-----------------------|-----------------|---------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| AircraftByID (int ID) | Aircraft object | --      | Retorna o objeto Aircraft para a ID específica, se esta existir; caso contrário, será nula. Por padrão, a aeronave do usuário sempre terá ID= 0. |
| CurrentFlight()       | Flight object   | --      | Retorna a informação sobre o voo atual.                                                                                                          |

## O Objeto Aircraft

O objeto Aircraft deixa você inspecionar e controlar aspectos de uma aeronave em particular, tais como a aeronave do usuário ou sua AI. O objeto Aircraft tem vários sub-objetos incluindo Autopilot, Radio e Fuel.

| Nome da Propriedade                          | Tipo da Propriedade | Unidade     | Atributos | Descrição/notas                                                                                                 |
|----------------------------------------------|---------------------|-------------|-----------|-----------------------------------------------------------------------------------------------------------------|
| AbsoluteAGLAltitude                          | double              | pés         | Recupera  | Altitude absoluta acima do nível do solo (AGL).                                                                 |
| AIController                                 | AIController Object | --          | Recupera  | Retorna o objeto <i>AIController</i>                                                                            |
| AileronTrim                                  | Signed Fraction     | -1.0 to 1.0 | Recupera  | -1.0 = asa esquerda toda baixa<br>+1.0= asa direita toda cima                                                   |
| AircraftID                                   | int                 | --          | Recupera  | Retorna a ID desta aeronave. A aeronave do usuário sempre terá ID = 0.                                          |
| AngleOfAttack                                | double              | graus       | Recupera  | Retorna o ângulo de ataque desta aeronave.                                                                      |
| Autopilot                                    | Autopilot object    | --          | Recupera  | Retorna o objeto Autopilot para essa aeronave                                                                   |
| Bank<br>(inclinação lateral de um aeroplano) | double              | graus       | Recupera  | Retorna o valor da inclinação desta aeronave em graus. Varia de -180 a 180. Inclinação positiva é para direita. |
| Cockpit                                      | Cockpit Object      | --          | Recupera  | Retorna o objeto Cockpit para acessar os instrumentos e controles da aeronave.                                  |

*Continua Objeto Aircraft*

|                                                |                 |             |          |                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------|-----------------|-------------|----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| GearType                                       | enum            | --          | Recupera | Tipo de trem de pouso da aeronave:<br>1-GEAR_SYSTEM_WHEELS<br>2-GEAR_SYSTEM_RETRACTABLE_WHEELS<br>4-GEAR_SYSTEM_FLOATS<br>8-GEAR_SYSTEM_SKIDS<br>16-GEAR_SYSTEM_SKIS<br>É possível para uma aeronave ter uma mistura de tipos diferentes de trem de pouso. Neste caso, o valor de retorno é a soma dos tipos naquela localização. |
| GravityLoad                                    | double          | G's         | Recupera | O valor da aceleração gravitacional que o piloto/aeronave está atualmente experimentando.                                                                                                                                                                                                                                         |
| Groundspeed                                    | double          | Nós         | Recupera | Retorna a velocidade de solo da aeronave em nós.                                                                                                                                                                                                                                                                                  |
| Heading                                        | double          | Graus       | Recupera | Retorna a direção real da aeronave em graus (não a variação magnética)                                                                                                                                                                                                                                                            |
| IndicatedAirspeed                              | double          | Nós         | Recupera | Retorna a velocidade indicada no ar da aeronave em nós                                                                                                                                                                                                                                                                            |
| IndicatedAltitude                              | double          | Pés         | Recupera | Altitude mostrada no altímetro                                                                                                                                                                                                                                                                                                    |
| LeftFlapPosition                               | fraction        | 0.0 to 1.0  | Recupera | Posição real dos Flaps da esquerda: 0= todo recolhido e 1 = todo baixo                                                                                                                                                                                                                                                            |
| NumEngines                                     | double          | 0-4         | Recupera | Retorna o número de motores nesta aeronave.                                                                                                                                                                                                                                                                                       |
| OnGround                                       | Boolean         | --          | Recupera | Retorna se a aeronave está atualmente no chão.                                                                                                                                                                                                                                                                                    |
| OverspeedWarningOn                             | Boolean         | --          | Recupera | Retorna se o aviso de excesso de velocidade está ligado.                                                                                                                                                                                                                                                                          |
| Pitch<br>(inclinação longitudinal da aeronave) | double          | Pés         | Recupera | Retorna a inclinação da aeronave em graus. Valor negativo é inclinação para baixo e positivo é para cima.                                                                                                                                                                                                                         |
| Position                                       | Position object | --          | Recupera | Retorna um objeto <i>LatLonAlt</i> com a posição da aeronave                                                                                                                                                                                                                                                                      |
| PressureAltitude                               | double          | Pés         | Recupera | Altitude acima da pressão padrão da superfície.                                                                                                                                                                                                                                                                                   |
| RightFlapPosition                              | fraction        | 0.0 to 1.0  | Recupera | Posição real dos Flaps da Direita: 0= todo recolhido e 1 = todo baixo                                                                                                                                                                                                                                                             |
| RudderTrim                                     | Signed Fraction | -1.0 to 1.0 | Recupera | -1.0 = Leme todo à esquerda<br>+1.0= Leme todo à direita                                                                                                                                                                                                                                                                          |
| StallWarningOn                                 | Boolean         | --          | Recupera | Retorna se o aviso de estol está atualmente ligado.                                                                                                                                                                                                                                                                               |
| Track                                          | double          | Graus       | Recupera | Retorna a direção magnética corrigida pelo vento da aeronave                                                                                                                                                                                                                                                                      |
| TrueAirspeed                                   | double          | Nós         | Recupera | Retorna a velocidade no ar real da aeronave em nós                                                                                                                                                                                                                                                                                |
| True MSLAltitude                               | double          | Pés         | Recupera | Altitude real acima do nível do mar (MSL).                                                                                                                                                                                                                                                                                        |
| VerticalSpeed                                  | double          | Pés/min     | Recupera | Retorna a velocidade vertical da aeronave em pés por minuto.                                                                                                                                                                                                                                                                      |



| Nome do Método                                       | Tipo de Retorno                | Descrição                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------|--------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fuel                                                 | Coleção de objetos Fuel        | Retorna uma coleção de objetos <i>fuel</i> , um por tanque. Pode ser 0-size (planador)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Gear                                                 | Coleção de objetos LandingGear | Retorna uma coleção de objetos <i>LandingGear</i> , um para cada conjunto.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| VORRadio(UInt32 Index)                               | Objeto VORRadio                | Retorna o objeto específico <i>VORRadio</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ADFRadio(UInt32 Index)                               | Objeto ADF Radio               | Retorna o objeto específico <i>ADFRadio</i> .                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| COMRadio(UInt32 Index)                               | Objeto COM Radio               | Retorna o objeto específico <i>COMRadio</i>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Transponder                                          | Coleção de objetos Transponder | Retorna uma coleção de objetos transponder. Flight Simulator 2004 inclui apenas 1.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Engine( int iNum )                                   | Objeto Engine                  | Retorna um indicador para um objeto <i>Engine</i> para o motor dado.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| GaugeDisplay( string strGaugeName, string strAction) | --                             | <p>Define um estado para um mostrador em particular. Pode ser usado para qualquer mostrador. A literal <i>strGaugeName</i> deve ter o mesmo nome do mostrador usado no arquivo <i>panel.cfg</i>.</p> <p>Valores válidos para <i>strActions</i> são:<br/>           "Hide" – não mostra o mostrador,<br/>           "Blink" – pisca numa frequência de 2Hz,<br/>           "Cover" – Coberto de cinza,<br/>           "Hilight", "Normal" or ""<br/>           Por exemplo<br/> <i>Ac.GaugeDisplay("Airspeed", "Blink");</i></p>                                                                                                                                                      |
| PanelFailure(int GaugeIdx, BOOL bFail)               | Boolean                        | <p>Recupera um mostrador <i>GaugeID</i> e um indicador booleano mostrando se este mostrador deve estar funcionando ou não. O valor de retorno é o estado prévio à falha. Este método define uma falha no mostrador ao contrário do sistema da aeronave. O parâmetro booleano útil para checar onde o instrutor pode definir uma falha, concertá-lo ou falhar outro.</p> <p>0 - Falha na Atitude<br/>           1 - Falha no Giro Direcional<br/>           2 - Falha na Velocidade Vertical<br/>           3 - Falha no Altímetro<br/>           4 - Falha na Velocidade do ar<br/>           5 - Falha no Coordenador de Curvas</p>                                                 |
| PanelDisplay(int GaugeIdx, int bDisplayGauge)        | Boolean                        | <p>Recupera um mostrador <i>GaugeID</i> e um indicador booleano mostrando se o mostrador deve aparecer ou não.</p> <p>2 – Piscando – Apaga o mostrador rapidamente. Continua apagando até que a função seja chamada novamente com outros valores.<br/>           1 - Normal – exibe o mostrador normalmente<br/>           0 - Apagado – apaga o mostrador</p> <p>O retorno é o valor prévio do <i>gaugeDisplay</i></p> <p>0 - Falha na Atitude<br/>           1 - Falha no Giro Direcional<br/>           2 - Falha na Velocidade Vertical<br/>           3 - Falha no Altímetro<br/>           4 - Falha na Velocidade do ar<br/>           5 - Falha no Coordenador de Curvas</p> |
| SystemFailure(INT iSystemID, BOOL bFail)             | Boolean                        | <p>Retorna uma ID representando o sistema da aeronave que falha e um indicador booleano mostrando se o sistema deveria falhar. Pelo fato de que o sistema pode interagir com múltiplos mostradores, esta função é separada do método <i>PanelFailure()</i>.</p> <p>0 – Falha no sistema de Vácuo<br/>           1 – Falha no sistema de Tubo do Pitot<br/>           2 – Falha no sistema elétrico</p>                                                                                                                                                                                                                                                                               |

## O Objeto Autopilot

| Nome da Propriedade | Tipo da Propriedade | Unidade           | Atributos       | Descrição                                                                                                                                                                                                                                                                                              |
|---------------------|---------------------|-------------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Airspeed            | double              | Nós               | Recupera/Ajusta | Se <i>AirspeedHold</i> for TRUE, o piloto automático fixa a velocidade assinalada.                                                                                                                                                                                                                     |
| AirspeedHold        | Boolean             | --                | Recupera/Ajusta | TRUE se é para o piloto automático manter a velocidade                                                                                                                                                                                                                                                 |
| Altitude            | double              | Pés               | Recupera/Ajusta | Se o retentor de altitude estiver acionado, este valor controla a altitude utilizada pelo piloto automático.                                                                                                                                                                                           |
| AltitudeHold        | Boolean             | --                | Recupera/Ajusta | Posição de seleção do retentor de altitude                                                                                                                                                                                                                                                             |
| ApproachHold        | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para seguir o localizador e o glide slope da estação sintonizada em NAV 1                                                                                                                                                                                 |
| ArmAutoThrottle     | Boolean             | --                | Recupera/Ajusta | TRUE quando o autothrottle estiver armado. Esta funcionalidade é específica da aeronave.                                                                                                                                                                                                               |
| AttitudeHold        | Boolean             | --                | Recupera/Ajusta | Posição de seleção do retentor de atitude                                                                                                                                                                                                                                                              |
| AutoBrake           | Boolean             | --                | Recupera/Ajusta | TRUE quando os freios são aplicados automaticamente.                                                                                                                                                                                                                                                   |
| BackCourseHold      | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para voar o back course da estação sintonizada em NAV 1.                                                                                                                                                                                                  |
| FlightDirectorOn    | Boolean             | --                | Recupera/Ajusta | TRUE quando o Flight Director estiver ligado.                                                                                                                                                                                                                                                          |
| GlideSlopeHold      | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para capturar o glide slope sintonizado em NAV 1.                                                                                                                                                                                                         |
| Heading             | double              | Graus (magnético) | Recupera/Ajusta | Se o retentor de direção estiver acionado, este valor controla a direção usada pelo piloto automático.                                                                                                                                                                                                 |
| HeadingHold         | Boolean             | --                | Recupera/Ajusta | Posição de seleção do retentor de direção                                                                                                                                                                                                                                                              |
| LocalizerHold       | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para seguir o localizador sintonizado em NAV 1.                                                                                                                                                                                                           |
| Mach                | double              | --                | Recupera/Ajusta | Se <i>MachHold</i> for TRUE, o piloto automático vai manter o valor numérico de Mach                                                                                                                                                                                                                   |
| MachHold            | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para manter a velocidade em Mach.                                                                                                                                                                                                                         |
| Máster              | Boolean             | --                | Recupera/Ajusta | Seletor principal do Piloto Automático. Este deve ser ajustado para TRUE para o piloto automático estar ativo e para que os outros ajustes dele também funcionem. Tipicamente, um script define os ajustes do piloto automático e então define este seletor principal para TRUE para iniciá-lo ligado. |
| NavHold             | Boolean             | --                | Recupera/Ajusta | TRUE se o piloto automático/Flight Director estiver ajustado para interceptar e rastrear a radial mostrada no OBI 1/HIS.                                                                                                                                                                               |

|               |         |            |                 |                                                                                                                                                                                                                                                |
|---------------|---------|------------|-----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| VerticalSpeed | double  | pés/minuto | Recupera/Ajusta | Se o retentor de altitude estiver acionado, este valor controla a velocidade vertical usada pelo piloto automático. Este valor é calculado pelo piloto automático quando o retentor de altitude estiver acionado. Pode ser mudado pelo script. |
| WingLeveler   | Boolean | --         | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para manter o nivelador de asas.                                                                                                                                                                  |
| YawDamper     | Boolean | --         | Recupera/Ajusta | TRUE se o piloto automático estiver ajustado para minimizar a quantidade de guinadas.                                                                                                                                                          |

## O Objeto Cockpit

Este objeto contém as propriedades para instrumentos e controles que um piloto tem de acessar quando sentado no assento de uma aeronave.

| Nome da Propriedade       | Tipo de Propriedade     | Unidade    | Atributos       | Descrição                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|-------------------------|------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EngineController(intiNum) | EngineController object | --         | Recupera        | Retorna um indicador para um objeto Engine Control. Este objeto permite ao script controlar a entrada de dados em um motor específico.                                                                                                                                                                                                                   |
| ElevatorTrim              | Signed Fraction         | -1.0 a 1.0 | Recupera/Ajusta | -1.0 = Nariz todo para baixo<br>+1.0 = Nariz todo para cima                                                                                                                                                                                                                                                                                              |
| FlapsHandleMax            | Int                     | --         | Recupera        | Retorna o número do valor da alavanca do Flap na aeronave atual. Este é o valor máximo permitido na propriedade <i>FlapsHandlePos</i> . Um valor de zero em <i>FlapsHandleMax</i> significa que Flaps não estão disponíveis nesta aeronave.                                                                                                              |
| FlapsHandlePos            | Int                     | --         | Recupera/Ajusta | A posição atual dos Flaps.<br>0 = totalmente recolhidos<br>1 = estendidos um estágio<br>2 = estendidos dois estágios.<br><br>O número mostra a correspondência na alavanca e a posição correspondente dos Flaps varia de aeronave para aeronave. A propriedade <i>FlapsHandleMax</i> retorna o número da posição da alavanca na aeronave correspondente. |
| IndicatedAltitude         | Double                  | Pés        | Recupera        | A altitude mostrada no altímetro.                                                                                                                                                                                                                                                                                                                        |
| KollsmanWindowSetting     | Double                  | Em mmHg    | Recupera/Ajusta | Valor da janela de Kollsman no altímetro. É a pressão de referência para o altímetro.                                                                                                                                                                                                                                                                    |
| LandingGearHandlePos      | Int                     | --         | Recupera/Ajusta | A posição do trem de pouso.<br>1 = recolhido<br>0 = abaixado<br><br>Ajustando essa propriedade muda a posição da alavanca e recolhe ou abaixa o trem de pouso.                                                                                                                                                                                           |
| LandingLightsOn           | Boolean                 | --         | Recupera/Ajusta | TRUE se as luzes de pouso estiverem acesas.                                                                                                                                                                                                                                                                                                              |



|                                          |                 |            |                     |                                                                                                                                                                                                             |
|------------------------------------------|-----------------|------------|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Marker                                   | Enum            | --         | Recupera            | Retorna um enum para os Marca-<br>dores:<br><i>Inner (IM)</i> : se o inner marker estiver<br>ligado.<br><i>Middle (MM)</i> – se o middle marker<br>estiver ligado<br><i>Outer (OM)</i> – se o outter marker |
| NAVGPSSwitch                             | Enum            | --         | Recupera/<br>Ajusta | Indica se os dados de navegação a<br>serem exibidos provêm de GPS ou<br>VOR .<br>0=NAV<br>1=GPS<br><br>Adquirindo ou ajustando um valor<br>TRUE indica que a navegação a<br>ser exibida provêm do GPS.      |
| PanelLightsOn                            | Boolean         | --         | Recupera/<br>Ajusta | TRUE se as luzes do painel estive-<br>rem acesas.                                                                                                                                                           |
| ParkingBrakeOn                           | Boolean         | --         | Recupera/<br>Ajusta | TRUE se os freios de estaciona-<br>mento estiverem acionados; caso<br>contrário, FALSE.                                                                                                                     |
| RudderPedalPos ( <i>pedais de leme</i> ) | Signed fraction | -1.0 a 1.0 | Recupera            | -1=tudo para esquerda,<br>0=centralizado,<br>1=tudo para direita                                                                                                                                            |
| SmokeSystemOn                            | Boolean         | --         | Recupera/<br>Ajusta | TRUE se smoke system estiver<br>ligado.                                                                                                                                                                     |
| StrobesOn                                | Boolean         | --         | Recupera/<br>Ajusta | TRUE se as luzes estroboscópicas<br>estiverem ligadas.                                                                                                                                                      |
| TurnCoordinatorNeedle                    | Signed Fraction | -1.0 a 1.0 | Recupera            | -1=toda para esquerda,<br>0=centrada,<br>1=toda para direita                                                                                                                                                |
| TurnCoordinatorBall                      | Signed fraction | -1.0 a 1.0 | Recupera            | -1=toda para esquerda,<br>0=centrada,<br>1=toda para direita                                                                                                                                                |
| YokeAileronPos                           | Signed fraction | -1.0 a 1.0 | Recupera            | -1=toda para esquerda,<br>0=centrada,<br>1=toda para direita                                                                                                                                                |
| YokeElevatorPos                          | Signed fraction | -1.0 a 1.0 | Recupera            | -1=toda para esquerda,<br>0=centrada,<br>1=toda para direita                                                                                                                                                |

### O Objeto EngineControl

| Nome da Propriedade | Tipo da Propriedade | Unidade    | Atributos       | Descrição                                                                                 |
|---------------------|---------------------|------------|-----------------|-------------------------------------------------------------------------------------------|
| CarburetorHeat      | Bool                | --         | Recupera/Ajusta | Ajusta o aquecedor do carburador em<br>ligado/desligado para esse motor                   |
| CowlFlaps           | fractional          | 0.0 to 1.0 | Recupera/Ajusta | Indica o grau no qual os cowl flaps estão                                                 |
| EmergencyPower      | Boolean             | --         | Recupera/Ajusta | TRUE quando a potência de emergência<br>está ativa para esse motor                        |
| Mixture             | float               | 0.0 to 1.0 | Recupera/Ajusta | O nível da posição do misturador para<br>esse motor.                                      |
| Prop                | Signed Fraction     | -1.0 a 1.0 | Recupera/Ajusta | O nível da posição do propulsor para<br>esse motor. Negativo para feathering.             |
| Throttle            | signed fraction     | -1.0 a 1.0 | Recupera/Ajusta | O nível da posição do acelerador para<br>esse motor. Negativo para aceleração<br>reversa. |

## O Objeto Engine

O objeto engine é um conjunto de propriedades *Recupera* que representam os valores reais neste motor. Para controlar o motor, o autor do script precisa ir para o objeto cockpit.engineControl e ajustar isso por lá.

| Nome da Propriedade | Tipo da Propriedade | Unidade | Atributos | Descrição                                                                                                                           |
|---------------------|---------------------|---------|-----------|-------------------------------------------------------------------------------------------------------------------------------------|
| EngineType          | enum                | --      | Recupera  | Id para os tipos de motores: 0 = Pistão<br>1 = Jato<br>2 = Sem motor (planador)<br>3 = Turbo-Hélice<br>4 = Foguete<br>5 = TurboProp |
| manifoldPressure    | double              | --      | Recupera  | A pressão principal para esse motor                                                                                                 |
| RPM                 | int                 | RPM     | Recupera  | A RPM para esse motor                                                                                                               |
| N1                  | Float               | %       | Recupera  | N1 RPM desta turbina (motor)                                                                                                        |
| N2                  | Float               | %       | Recupera  | N2 RPM desta turbina (motor)                                                                                                        |

| Nome do Método | Tipo de Retorno | Unidade | Descrição                                                                                                        |
|----------------|-----------------|---------|------------------------------------------------------------------------------------------------------------------|
| Fail( BOOL )   | --              | --      | Permite ao script disparar ou reparar uma falha neste motor. O parâmetro Booleano indica se o motor deve falhar. |

## O Objeto Combustível

| Nome da Propriedade | Tipo da Propriedade | Unid   | Atributos           | Descrição                                                                                                                                                                                                                                                                                  |
|---------------------|---------------------|--------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Capacity            | float               | galões | Recupera            | Retorna a capacidade total de todos os tanques na aeronave, em galões.                                                                                                                                                                                                                     |
| PercentFull         | float               | --     | Recupera/<br>Ajusta | A porcentagem da capacidade total que está cheio, por todos os tanques. Se esta propriedade for definida, todos os tanques terão igualmente combustível com a porcentagem especificada.                                                                                                    |
| FuelType            | enum                | --     | Recupera            | Retorna um índice indicando o tipo de combustível:<br>0 = NONE<br>1 = AVGAS<br>2 = JETFUEL<br>3 = ROCKET                                                                                                                                                                                   |
| Location            | enum                | --     | Recupera            | Retorna uma ID indicando a localização do tanque:<br>0 = Principal Direito<br>1 = Principal Esquerdo<br>2 = Auxiliar Direito<br>3 = Auxiliar Esquerdo<br>4 = Ponteiro Direito<br>5 = Ponteiro Esquerdo<br>6 = Central<br>7 = Central 2<br>8 = Central 3<br>9 = Externo 1<br>10 = Externo 2 |

### O Objeto Rádio VOR

| Nome da Propriedade | Tipo da Propriedade | Unidade           | Atributos       | Descrição                                                                                                                                                                                                                  |
|---------------------|---------------------|-------------------|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Frequency           | String              | MHz               | Recupera        | A frequência sintonizada e ativa no radio VOR                                                                                                                                                                              |
| StandbyFrequency    | String              | MHz               | Recupera/Aju    | A frequência de reserva no radio.                                                                                                                                                                                          |
| Active              | Boolean             | --                | Recupera        | Retorna TRUE se o radio é sintonizado em uma estação em alcance. Se for um objeto VOR será TRUE se a estação for VOR e estiver ativa. Para rádios ADF, será TRUE se uma estação ADF for sintonizada.                       |
| StationLatLon       | Position object     | --                | Recupera        | A posição da estação sintonizada. Retorna nulo se nenhuma estação for sintonizada.                                                                                                                                         |
| DMEDistance         | float               | Metros            | Recupera        | A distância em metros à estação sintonizada. Retorna 0 se nenhuma for sintonizada.                                                                                                                                         |
| StationIdent        | string              | --                | Recupera        | O identificador ICAO da estação sintonizada.                                                                                                                                                                               |
| ToFrom              | int                 | --                | Recupera        | Retorna 0 se nenhum VOR estiver sintonizado; 1 se na direção To para um VOR sintonizado e 2 se na direção From da estação.                                                                                                 |
| BackCourse          | Boolean             | --                | Recupera        | Retorna TRUE se este for um back-                                                                                                                                                                                          |
| Radial              | integer             | Graus (magnético) | Recupera        | Radial do VOR sintonizado; 0 se nenhuma estação sintonizada.                                                                                                                                                               |
| OBS                 | int                 | Graus             | Recupera/Ajusta | Valor do OBS (Omni Bearing Select) colocado no VOR                                                                                                                                                                         |
| LocalizerAvailable  | Boolean             | --                | Recupera        | TRUE se a estação VOR for uma estação ILS                                                                                                                                                                                  |
| Localizer           | integer             | Graus (reais)     | Recupera        | Curso do Localizer indicado pelo VOR; 0 se o localizador não estiver ativo.                                                                                                                                                |
| GSAvailable         | Boolean             | --                | Recupera        | TRUE se o ILS fornecer glide slope.                                                                                                                                                                                        |
| GSAvailable         | Boolean             | --                | Recupera        | Retorna TRUE se o glide slope estiver ativo                                                                                                                                                                                |
| GSDeviation         | float               | --                | Recupera        | A posição de afastamento da agulha do glide slope no indicador VOR, em variação de -1.0 a 1.0; -1.0 significa que a agulha está no topo de sua variação e 1.0 no fundo. Retorno 0 significa que não há glide slope ligado. |
| CourseDeviation     | float               | --                | Recupera        | O curso de desvio do VOR sintonizado. Isto retorna um valor entre -180.0 a 180.0 graus, onde 180.0 significa todo à esquerda, 180.0 todo à direita e 0 centrado em TO ou nenhum VOR sintonizado.                           |
| DMEAvailable        | Boolean             | --                | Recupera        | TRUE se o VOR sintonizado oferecer DME                                                                                                                                                                                     |



| Nome do Método    | Tipo de Retorno | Unidade | Descrição                                                                                                                                                                                                               |
|-------------------|-----------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Fail(enum System) | --              | --      | Permite ao script especificar uma falha no rádio:<br>failWhole (falhar em tudo)<br>failAudio (falhar Áudio)<br>failVOR (falhar VOR)<br>failGlideSlope (falhar GS)<br>failLocalizer (falhar LOC)<br>failDME (falhar DME) |
| Flipflop          | --              | --      | Permite ao script escolher entre frequências ativas e em espera neste rádio.                                                                                                                                            |

### O Objeto Trem de Pouso

| Nome da Propriedade | Tipo da Propriedade | Unid    | Atributos | Descrição                                                                              |
|---------------------|---------------------|---------|-----------|----------------------------------------------------------------------------------------|
| Position            | fraction            | 0.0-1.0 | Recupera  | A posição do trem de pouso:<br>0 = todo recolhido<br>1 = todo baixo                    |
| Location            | enum                | --      | Recupera  | Localização:<br>0 = Centro<br>1 = Esquerda<br>2 = Direita<br>3 = Cauda<br>4 = Auxiliar |

### O Objeto Rádio COM

| Nome da Propriedade | Tipo da Propriedade | Unid | Atributos          | Descrição                                            |
|---------------------|---------------------|------|--------------------|------------------------------------------------------|
| frequency           | String              | MHz  | Recupera           | A frequência que está ativa no radio de comunicação. |
| StandbyFrequency    | String              | MHz  | Recupera/<br>Ajuta | A frequência de espera neste rádio                   |

| Nome do Método    | Tipo de Retorno | Unidade | Descrição                                                                                                        |
|-------------------|-----------------|---------|------------------------------------------------------------------------------------------------------------------|
| Fail(enum System) | --              | --      | Permite ao script especificar uma falha neste rádio: <i>failWhole</i> , <i>failTransmit</i> , <i>failRecieve</i> |
| FlipFlop          | --              | --      | Permite ao script alternar entre as frequências ativa e em espera.                                               |

### O Objeto de Rádio Transponder

| Nome da Propriedade | Tipo da Propriedade | Unidade | Atributo           | Descrição                                                               |
|---------------------|---------------------|---------|--------------------|-------------------------------------------------------------------------|
| TransponderCode     | integer             | --      | Recupera/<br>Ajuta | Código do transponder em 4 dígitos. Cada dígito deve estar entre 0 - 7. |

| Nome do Método    | Tipo de Retorno | Unidade | Descrição                                                                                                            |
|-------------------|-----------------|---------|----------------------------------------------------------------------------------------------------------------------|
| Fail(enum System) | --              | --      | Permite ao script especificar uma falha neste rádio:<br>failWhole (falhar tudo)<br>failTransmit (falhar Transmissão) |

## O Objeto de Rádio ADF

| Nome da Propriedade | Tipo da Propriedade | Unidade | Atributos       | Descrição                                                                      |
|---------------------|---------------------|---------|-----------------|--------------------------------------------------------------------------------|
| Frequency           | String              | MHz     | Recupera/Ajusta | A frequência em que esse ADF está sintonizado.                                 |
| Active              | Boolean             | --      | Recupera        | Retorna TRUE se uma estação ADF está sintonizada                               |
| StationLatLon       | Position object     | --      | Recupera        | A posição da estação sintonizada. Retorna nulo se não há estação sintonizada.  |
| StationIdent        | string              | --      | Recupera        | O identificador ICAO da estação sintonizada.                                   |
| Bearing             | float               | Graus   | Recupera        | O rumo real para a estação ADF. Retorna 0 se nenhuma estação está sintonizada. |

| Nome do Método    | Tipo de Retorno | Unidade | Descrição                                                                                                                                           |
|-------------------|-----------------|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| Fail(enum System) | --              | --      | Permite ao script especificar falhas para esse rádio<br><br>failWhole (falhar tudo)<br>failAudio (falhar Áudio)<br>failSenseAntenna (falhar Antena) |

## O Objeto AIController

O objeto AIController contém um conjunto de métodos que permitem ao script fornecer ao AI aircraft um propósito.

| Nome do Método | Tipo de Retorno | Unidade | Descrição                                                                                                                                                                                                   |
|----------------|-----------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| StartEngine()  | --              | --      | StartEngine() inicia os motores da aeronave. Pense nele como uma forma de auto-inicialização. Este método funciona como um gatilho; chame-o uma vez para os motores iniciarem e novamente para desligá-los. |
| Pushback()     | --              | --      | PushBack() só é válido se a aeronave estiver estacionada num portão. Esse método faz com que o AIcontroller empurre a aeronave pra trás, saindo do portão.                                                  |

## O Objeto Flight

Este objeto reúne as informações sobre o usuário de voo atual. Ele fornece controle de alto nível sobre a situação, ou modo, de como o usuário está participando do mundo na simulação.

| Nome da Propriedade | Tipo da Propriedade | Unidade  | Atributos | Descrição                                                   |
|---------------------|---------------------|----------|-----------|-------------------------------------------------------------|
| ElapsedTime         | int                 | Segundos | Recupera  | Retorna o número de segundos desde que o voo foi carregado. |

| Nome do Método           | Tipo de Retorno | Unidade | Descrição                                                                                                                                       |
|--------------------------|-----------------|---------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| Load(string StrFileName) | Boolean         | --      | Se <i>strFileName</i> é nulo, exibe a caixa de diálogo Load Flight, carrega o voo específico e retorna TRUE se o voo é carregado adequadamente. |
| Save()                   | --              | --      | Envia o comando de menu de salvar. Isto faz com que o diálogo "save a flight" apareça e permite ao usuário especificar um novo nome de arquivo. |

|                                                            |         |    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------------------------------------|---------|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SetScriptState(<br>String strProperty,<br>string strValue) | --      | -- | Este método pega um par de valores nominais e os salva num “pacote” que poderá ser usado por todo um script para gravar informações sobre um determinado estado. Isto é potencialmente útil quando você quer revisar ou retomar uma condição prévia.<br>Ao atingir o final de cada missão, um script com o estado de suas variáveis atual poderá ser salvo, permitindo ao usuário retornar ao local desejado usando UserEvents.<br>O conteúdo deste “pacote” também será escrito no arquivo de vôo do usuário quando este salvar o vôo. Isto permitirá ao usuário salvar seus vôos após ter atingido seu objetivo e (caso a aventura tenha sido escrita apropriadamente) estar apto a retornar àquele estado quando o arquivo for carregado novamente. |
| getScriptState(<br>string strProperty)                     | String  | -- | Definida uma propriedade, esta função procura pelo último valor ajustado para ela. Se o vôo foi previamente salvo, então o valor retornado será obtido do que estará salvo no arquivo de vôo.<br>No evento Init(), o script poderá investigar o método <b>getScriptState</b> procurando por alguma propriedade conhecida que pode ter sido salva num evento <b>OnFlightSave()</b> prévio e usar isto para recuperar o script com o estado do vôo e da aeronave.<br>Ele retorna uma string vazia se não houver nenhum valor salvo no nome da propriedade.                                                                                                                                                                                               |
| Reset()                                                    | Boolean | -- | Reinicia todas as informações de flight/world. Retorna TRUE se o vôo é reiniciado corretamente.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| FlightAnalysis()                                           | --      | -- | Exibe caixa de diálogo de Análise do Vôo (modal) e pausa a simulação.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| End(number uExitCode                                       | --      | -- | Enfileira uma requisição para retornar ao menu inicial do jogo; <i>uExitCode</i> comunica a saída da lição<br><br>0 = sucesso,<br>1 = falha e<br>2 = apresentação do certificado.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |

## Eventos

Eventos em ABL não são tão parecidos com os scripts de outras linguagens. Pelo fato de ABL não ser uma linguagem orientada por objeto, esta não pode exportar funções para serem chamadas em respostas à eventos.

De modo geral, eventos em ABL são definidos como *palavras-chaves reservadas* na linguagem. Nomes de funções carregadas automaticamente quando o script principal dispara-as. Como resultado, esses eventos podem ser considerados globais. A lista abaixo representa nomes de funções reservadas chamadas automaticamente do Flight Simulator.

| Nome do Evento | Descrições/Notas                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Init()         | Chamado quando o script é inicializado (quando o vôo é carregado)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| OnFlightSave() | Chamado quando o usuário salva um vôo com um script anexado. Isto é novo no FS2004. Este é chamado depois de pressionado OK na caixa de diálogo de salvar o vôo e antes do vôo ser escrito no disco. O script vai ter a oportunidade de colocar qualquer estado ou valores de variáveis no pacote da propriedade scriptState no objeto do vôo. Qualquer par de valores nominais neste pacote irá permanecer no disco no arquivo de vôo novo e o script terá a chance de ler estes scripts durante o evento Init na próxima vez em que o vôo for carregado. Veja Modelos de Objetos do FS para maiores detalhes. |
| OnScriptExit() | Chamado quando o script está próximo de ser fechado (quando o vôo está terminando)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| OnCrash()      | Chamado quando a aeronave colide                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| OnStall()      | Chamado quando a aeronave entra em estol                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| OnOverspeed()  | Chamado quando a aeronave excede a velocidade                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| OnUserEvent1() | Chamado quando o usuário dispara a tecla 1 de sequência de eventos do script                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| OnUserEvent2() | Chamado quando o usuário dispara a tecla 2 de sequência de eventos do script                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

*OnUserEvent1* e *OnUserEvent2* foram incluídos para permitir ao script consultar o usuário para a entrada para de um estado avançado do script – por exemplo para avançar uma lição para o próximo passo ou para repetir um componente de uma lição.



# Criando seu primeiro script

Um script ABL no Flight Simulator é carregado quando um arquivo de Voo (.flt) faz referência a ele. O script para ser carregado deve ser especificado na seção [Main] do arquivo de voo. Portanto, antes de mais nada, crie seu voo. Somente através dele é que será possível você carregar o script. O script funciona como o esqueleto, a espinha dorsal do voo que possui uma aventura ou lição. O voo sem o script é carregado sem problema algum mas não tem como você carregar um script sem um voo, pois o script que vai dar forma e sustentação na aventura ou lição do voo que você está criando. Ao invés de criar um



plano de voo, crie simplesmente um voo simples: escolha o local de partida — pode ser na pista ou estacionado, o clima, a hora do saída, a estação e a aeronave desejada. Vamos chamar esse voo de modelo01. Clique em *Save Flight* e nomeie como modelo01 e coloque em description uma breve descrição sobre esse voo, tipo “voo de treinamento em ABL” ou qualquer coisa do tipo. Após feito isso, vá à sua pasta MEUS DOCUMENTOS\FLIGHT SIMULATOR FILES e procure pelo arquivo **modelo01.FLT** e abra-o com o bloco de notas (p.ex.: *botão direito do mouse sobre ele no Windows Explorer > abrir com > bloco de notas ou EditPad Lite*). Provavelmente você vai encontrar o seguinte nas linhas iniciais:

```
[Main]
Title=modelo01
Description=
AppVersion=9.0.30612
FlightVersion=1
```

[Tower]

*Continua.....*

Você deve incluir a chamada do arquivo abl em qualquer posição na seção [main] da seguinte maneira:

```
[Main]
Title=modelo01
ablscript=modelo01
Continua....
```

Lembre-se: **não inclua a extensão abl**, somente o nome do arquivo, neste caso considerando que o script vai ficar no arquivo **modelo01.abl**. Caso o script esteja numa pasta diferente do arquivo de voo, inclua o caminho também. Como sugestão, vou descrever o que sempre faço: crie uma pasta em [raiz]\Flights com o nome que você quiser, neste caso use, por exemplo, MODELOS. Para dentro dessa pasta recém criada traga os arquivos **modelo01.flt** e **modelo01.WX**. Dentro dessa mesma pasta é o local ideal para você criar seu script. Aproveitando que você está com o EditPad Lite aberto, onde você modificou seu arquivo de voo, crie um novo arquivo (*arquivo > novo*) e salve-o como (*arquivo > salvar como*) **modelo01.abl** na pasta MODELOS. Coloque o nome com a extensão ao salvar em *Nome do Arquivo* e não esqueça de, na linha abaixo em *Salvar como tipo*, colocar *Todos (\*.\*)*. Pronto, a matriz para criação de seu primeiro script está pronta e seu arquivo de voo configurado para chamar o script.

Vamos criar um exemplo bem básico inicialmente pra você ver progressivamente como a coisa funciona. Em primeiro lugar, é sempre válido colocar anotações por todo script pra saber o que está fazendo. Comece então com um cabeçalho, utilizando aqueles recursos de anotações já discutidos anteriormente: ou */\* \*/* entre o texto, ou *//* no início de cada linha.

```
/*
*****
Descrição: Script modelo01
Autor:
Data: 18.12.2003
Qualquer coisa mais.....
*****
*/
```

Feito isso, agora colocaremos as estruturas básicas e necessárias do módulo. Usarei cores diferentes para as diferentes seções somente para facilitar a compreensão:

```
#debug_on
#print_on

module modelo01;

Const
Var
Code

endmodule;
```

Temos as diretivas debug e print na condição ON para que possamos ter a janela tipo DOS para observarmos como o script está sendo executado, caso tenham erros, quais e onde esses estarão. Após tudo terminado, é só colocar *//* na frente de cada um para desabilitá-los, quando for distribuir sua aventura ou lição.

## Como scripts ABL são executados

O script é carregado e analisado com a inicialização do voo. Se não houver erros detectados, as variáveis globais e as constantes serão alocadas e definidas. O mecanismo do script imediatamente dispara o evento *Init()*, que resulta na execução do código dentro da função *Function Init()*, se existir para ser executado. Esta é a oportunidade para o script realizar uma inicialização primária das variáveis globais. Note que a simulação ainda está iniciando quando o ponto de entrada é chamado. Algumas variáveis da simulação podem não estar estáveis por vários segundos após o loop principal iniciar. O script é responsável por determinar seu próprio estado e executar ramificações apropriadas.

A árvore **SWITCH/CASE** é uma excelente maneira de se trabalhar com essa situação. O uso criterioso dessa construção permite quebrar script o em uma série de scripts menores e semi-independentes. Um script ABL é executado normalmente a uma taxa de uma vez por segundo de tempo da simulação. Deste ciclo, um script pode estimar o tempo gasto simplesmente contando quantas vezes foi executado. Essa execução uma-vez-por-segundo (ops) também pode ser usado como uma constante de integração. Por exemplo, transformar taxa em graus por segundo pode ser estimada determinando a diferença entre o curso presente e o curso do último ciclo. Se o presente curso é 030° e era 020° um segundo atrás, a taxa de conversão deve ser de 10° por segundo. Entretanto, se um controle mais preciso sobre o tempo for necessário, *t* é mais preciso e sólido para usar o modelo de objeto exposto. Se o voo está sendo executado em uma máquina particularmente lenta, com MUITO POUCA taxa de quadros (*frame*), é possível que o mecanismo de funcionamento do script seja executado em intervalos inconsistentes. O objeto *flight*, mostrado acima, expõe uma propriedade *elapsedTime* que retorna o número de segundos desde que o voo foi carregado, que é independente da taxa de quadros ou da performance da máquina.

O loop principal do script deve determinar seu próprio estado e o controle de execução apropriadamente. O script ABL continua e executar como parte do canal principal do simulador até que o voo termine ou um erro no ABL ocorra. ABL não inclui uma função de parar ou encerrar uma execução. Você é responsável por detectar que a aventura terminou e por escrever um script que responda de acordo.



```

// 3) Função de inicialização // Define dados importantes para execução da aventura / lição
function Init;
var
code
// Variáveis globais para inicializações das lições. Define os objetos básicos
S = getShell; // Objeto Shell
S.PlaySound("",0,0); // Som nulo
Writeln("Initializing"); // Mensagem de inicialização
wTheWorld = getWorld; // Objeto World
acMe = wTheWorld.AircraftByID(0); // Objeto Aeronave
AP = acMe.Autopilot; // Objeto Piloto Automático
fltThisFlight = wTheWorld.CurrentFlight; // Voo atual.

// Define os membros dos objetos básicos
cpOffice = acMe.Cockpit; // Objeto Cabine (cockpit)

// Variáveis de Tempo
nElapsedTime = 0; // Zero tempo inicial
nTimer = 0; // Zero cronômetro

// Outras coisas
nState = CaseInit; // Inicia os casos no COde em CaseInit
strWavBase = "Flights\\Modelos\\wav\\"; // Pasta onde ficam os arquivos de som
bUserKey1 = FALSE; // Tecla de atalho 1: CTRL + [

// Mensagens iniciais:
S.TextMessage(mensg01,1); // Exibe mensagem mensg01 de texto em amarelo

endfunction; // Encerra a função Function Init e inicia o código

Code // Início do código
nElapsedTime = fltThisFlight.ElapsedTime; // Define o tempo decorrido de simulação desde o início

switch (nState) //Aqui começamos "de verdade" com a aventura, com a definição dos estados.
case CaseInit: // CaseInit: Retarda em 10 segundos o início da aventura...
Writeln ("CaseInit"); // Escreve na janela o estado atual
if (nElapsedTime > 10) then // Caso o tempo transcorrido desde o início da simulação seja maior do que 10 segundos...
nState = CaseInicial; // O estado passará para o CaseInicial
endif; // Encerra a instrução IF
endcase; // Encerra o caso CaseInit

case CaseInicial: // Caso Inicial: Exibe mensagem 2.
Writeln("CaseInicial"); // Imprime na janela CaseInicial...
S.TextMessage(mensg02,0); // Mensagem 2 // Exibe na tela do simulador a mensg02 em branco
nState = CaseFinal; // Passa o estado para CaseFinal
endcase; // Encerra esse caso

case CaseFinal: // Case Final: Exibe a mensagem 03 em vermelho...
if (nElapsedTime > 30) then // ...após transcorridos 30 segundos do início da simulação
Writeln("CaseFinal"); //Escreve CaseFinal na janela...
S.TextMessage(mensg03,2); //...exibe a mensagem 03...
nState = CaseEncerra; //...muda o estado para CaseEncerra
endif; // Fim da instrução
endcase; // Fim do caso

case CaseEncerra: // CaseEncerra: aguarda o acionamento pelo usuário das teclas CTRL+[ para que a aventura
If (bUserKey1) then // se encerre e a janela de análise de voo apareça. Se você pressionar CTRL + [, então...
bUserKey1 = FALSE; //...teremos a linha de instrução liberando o comando da tecla...
fltThisFlight.FlightAnalysis; //...para que a janela de análise se abra após o...
fltThisFlight.End(nDoneFlag); //...encerramento do voo.
endif; // Fim da instrução
endcase; // Fim do caso

endswitch; // Fim da árvore SWITCH / CASE

endmodule. // Fim do módulo.

```

Neste primeiro exemplo, tivemos uma situação onde tivemos três mensagens de texto exibidas na tela e a utilização de uma combinação de teclas. Esse script é bem básico e já contém informações que serão úteis para todos as suas aventuras. Por isso, dê uma olhada boa nela e entenda o que está ocorrendo. Informações sobre cada instrução estão sendo dadas ao final de cada linha.

Este exemplo, assim como o a seguir — após as orientações a respeito de erros — virão juntamente compactados com esse guia. A maneira como ele foi escrito no exemplo acima é a mesma de como ele vai ficar no script que vai rodar a aventura no simulador.

Caso você não encontre os modelos prontos, pode tentar encontrá-los em [www.ivan.med.br/exemplos.zip](http://www.ivan.med.br/exemplos.zip). Se não conseguir encontrar nesse endereço, me mande um email para o endereço [maxflight@iva.med.br](mailto:maxflight@iva.med.br) que eu mando os arquivos.

O segundo exemplo é bem grande e completo com tudo o que vai precisar de modelo pra criar todas suas lições e aventuras.

Antes disso, vamos discutir alguma coisa sobre mensagens de erros, tendo sido todo o texto traduzido do guia de desenvolvimento em inglês na versão 2004.



## Como ABL informa erros

Depurar o script só é possível se a diretiva **#DEBUG\_ON** estiver presente no script. Uma janela DOS é criada (se esta já não existir) e as informações de depuração são exibidas lá. A primeira vez em que a janela de depuração é criada, o Flight Simulator entra em pausa (se o simulador estiver configurado em Settings/General em "Pause on Task Switch". NT: Geralmente eu configuro meu FS para operar com essa função desabilitada.).

Quando o script é carregado, o interpretador rastreia o script por erros na sintaxe e problemas como variáveis não declaradas, funções perdidas, etc. Se alguns desses erros forem detectados, ou se um erro de execução ocorrer, informações sobre o(s) erro(s) será(ão) mostrada(s) na janela de depuração. Erros de linguagem (i.e., não lógico) fazem o script interromper a execução. É importante notar que o voo, entretanto, não pausa ou termina.

A diretiva **#PRINT\_ON** envia instruções lógicas de depuração para a janela de depuração (via instrução **PRINT**). Para limitar o resultado de depuração de um seguimento do script, você pode controlar o resultado com a diretiva **#PRINT\_OFF**.

Sem ter uma diretiva habilitada, erros de sintaxe vão evitar que o script seja carregado e o único retorno para o usuário será uma caixa de mensagem informando ao usuário que a 'the adventure is corrupted.' Escolhendo OK faz com que você continue o voo, mas o script não será executado.

## Mensagens de Erro

Esta seção explicará algumas das mensagens de erro que você poderá encontrar, incluindo suas causas e como corrigi-las. Erros de mensagens geralmente incluem um nome de variável ou palavras reservadas. O erro estará próximo da variável/palavra, não necessariamente nela. Por exemplo, o código:

```
Print (y)
```

```
Print (" :");
```

1) Gera a seguinte mensagem de erro:

```
SYNTAX ERROR c:\Program Files\Microsoft Games\FS2002\FLIGHTS\myflts\badcode [line2]
-- (type 15) Unexpected token "Print"
```

O erro foi causado pela falta do ponto-e-vírgula na linha 1. Como em ABL você pode usar várias linhas de texto, o

interpretador realmente lê:

```
Print (y) Print (" :");
```

Adicionando um ponto-e-vírgula ao fim da primeira linha resolve o problema.

2) Alguns simples erros geram uma ou mais mensagens de erro. Se, por exemplo, a variável *nAVar* não é encontrada no script, o ABL informa a seguinte mensagem de erro:

**Undefined identifier "nAVar"**

**Incompatible types "nAVar"**

ABL pede que todas as constantes numéricas tenham um número, + ou - como seu primeiro caractere. Por exemplo, *x = .1*; gera erro; *x = 0.1* não.

3) ABL pede parênteses quando usar uma constante negativa:

```
X = X -- 1;
```

Cause:

**Invalid expression "x"**

**Incompatible types "x"**

**Incompatible assignment "x"**

Corrija esse problema usando:

```
X = X - (- 0.1);
```

4) Usando um '=' (atribuição) ao invés de '=' (comparação):

```
If (x = 5) then [correto aqui (x==5)]
```

**Return;**

**Endif;**

As mensagens de erro serão:

**Missing Right Parenthesis**

**Missing Then**

**Unexpected token "x"**

5) O erro oposto, usando '=' ao invés de '==':

```
X == 1; [correto aqui: x=1;]
```

Produz:

**Missing equal "x"**

**Invalid expression "x"**

**Incompatible types "x"**

## Erros de Sintaxe/ Interpretação

**Not a constant identifier "x"**  
(Identificador não-constante "x")

Os valores nos casos da árvore **SWITCH/CASE** devem ser constantes. Estes devem ser definidos por nomes constantes.

**Undefined identifier "z"**  
(Identificador indefinido "z")

Uma variável está sendo usada e não foi criada ou alocada na seção *Var*. Este é um erro de composição frequentemente encoberto.

**Incompatible assignment "nConstant"**  
(Atribuição incompatível "nConstant")

O programa está tentando atribuir um valor a uma constante ou uma variável que foi definida como de um tipo mas recebeu a atribuição de um outro tipo. Por exemplo, um *number* não pode ter um atribuído um valor de *string*. Um objeto *World* não pode receber a atribuição de um valor *number*.

## Erros ocultos

Esta seção descreve erros lógicos e outros tipos de erros "escondidos" nos scripts.

### Objeto Não Utilizado

Um dos erros mais frustrantes em programação é causado pela tentativa de acessar um objeto quando a variável não foi iniciada para aquele objeto (programadores de C/C++, pensem em um ponteiro não inicializado). Isso geralmente é causado pela negligência de se inicializar a variável de um objeto. A variável simplesmente não é atribuído valor. Considere o seguinte:

```
Var
Static Aircraft acMe;
Static World wTheWorld;
VORRadio VHFNAV1;
```

```
VHFNAV1 = acMe.VORRADIO(1);
Print (VHFNAV1.Frequency);
```

Nenhuma mensagem de erro é gerada, mas nada é exibido. Adicionando

```
acME = w.AircraftByID(0);
```

antes de:

```
VHFNAV1 = acMe.VORRADIO(1);
```

corrige o problema

### Discordância entre instruções IF/ THEN e CASE/ENDCASE

Um outro par de problemas para escrever o código é causada por desacordos entre instruções **IF/THEN** ou instruções **CASE/ENDCASE**. Frequentemente o erro da mensagem se refere à linha que está bem além do erro real ou dúzias de mensagens são geradas. Obviamente, a primeira coisa a checar é a última coisa modificada ou adicionada. Um **IF** não comparado, causa uma mensagem de erro do tipo "Missing ENDCASE" (**ENDCASE** faltando).

```

/*****

```

```

*   FILENAME:   Private Pilot CheckRide.abl
*   Prova de certificação para Piloto Privado do FS2004
*****/

```

**Descrição do arquivo abl**

```

#debug_on
#print_on

module modelo02;

const

```

**Detalhes de descrições já vistas  
no exemplo 1 não serão  
repetidos neste exemplo 2**

```

////////////////////////////////////
//                               //
////////////////////////////////////

```

```

//      Altitudes e Rumos para essa aventura

```

```

kAlt1      = 2000;
kAlt2      = 1300;
kAlt3      = 1500;
kAlt4      = 1680;
kAlt5      = 1980;
kAlt6      = 1750;
kAlt7      = 1550;
kHdg1      = 090;
kHdg2      = 180;
kHdg3      = 090;
kHdg4      = 000;

```

```

//      Itens que irão mudar dependendo da aeronave

```

```

bIsRecrip      = TRUE;      // Hélice?
bIsTurboProp   = FALSE;     // Turboprop?
bIsTurbine      = FALSE;    // Somente turbinas
bConstSpeedProp = FALSE;    // Prop de veloc constante?
nV1            = 40;        // V1
nVr            = 48;        // Vr
nV2            = 65;        // V2
bRetract       = FALSE;    // Trem de pouso retrátil?
nVX            = 70;        // Melhor ângulo de subida
nVY            = 75;        // Melhor taxa de subida
nVYSE          = 75;        // Best Single Engine Rate of Climb Velocity
nVNE           = 150;       // Velocidade nunca deverá exceder
nVMO           = 150;       // Velocidade operacional máxima
nMMO           = 0.210;     // Máx Mach operacional
nVLRC          = 120;       // Velocidade de cruzeiro
nCeiling       = 15000;     // Altitude Máxima certificada
nNormAlt       = 9500;      // Altitude normal de cruzeiro

```

```

//      Constantes específicas para essa lição

```

```

kRwy35Lat      = 47.2611;   // KTIW Runway 35 lat
kRwy35Lon      = -122.57934; // " " " lon
kSin2          = 0.034899496703; // sin 2 degrees
kSin3          = 0.052335956243; // sin 3 degrees
kSin4          = 0.069756473744;
kNmIFt         = 6076;      // 1 nautical mile = 6030 feet
kTDZE          = 292;       // Runway touchdown zone elevation

```

```

// Conversão

```

```

Mtrs2Ft        = 3.28084;   // Metros para pés
Ft2Mtrs        = 0.3048;    // Pés para metros

```

```

// Definições

```

```

defLeft         = 0;        // Esquerda
defRight        = 1;        // Direita
nSQLength       = 50;       // Número máximo de arquivos de som que podem ser enfileirados
SQ_PLAY         = 0;        // Toca um som
SQ_PAUSE        = 1;        // Mostra algum texto
SQ_STATE        = 2;        // Mostra algum texto
SQ_MESSAGE      = 3;        // Mostra algum texto
SQ_GAUGEHLIGHT  = 4;        // Destaca um mostrador
SQ_THROTTLE     = 5;        // Ajusta a velocidade
SQ_RESETTIMER   = 6;        // Reinicia o cronômetro
SQ_TOLERANCE    = 7;        // Aciona Checagem de Tolerância
SQ_TOLERANCEFLAG = 8;
THROTTLE_MANUAL = 0;        // Aceleração sob controle manual
THROTTLE_LEVEL  = 1;        // Aceleração fixa em porcentagem constante
THROTTLE_RPM    = 2;        // Aceleração fixa em RPM constante
// ..... continua

```

**Todas essas constantes são exclusivas para essa aventura, não servindo nem para as outras que já vem com o FS2004. Portanto, podem ser deixadas de lado. Você até pode incluí-las em suas aventuras alterando seus valores.**

**A Conversão e as Definições são utilizadas por outras aventuras e vôos do FS2004 e principalmente definições têm conteúdo bastante útil para suas lições, principalmente nas funções. Vale a pena entender com o decorrer do exemplo onde elas se encaixarão e por isso vale a pena mantê-las.**

**Devido ao fato de que algumas listas de constantes e variáveis são muito grandes e desnecessário descrever todos seus itens, terão seu conteúdo resumido com a expressão:**

//..... continua

**Isto só é valido aqui nas descrições. Nos exemplos, estes estarão completos.**

```

///
//                                     Constantes para as Árvores SWITCH / CASES                                     //
///

CasePreFlight  = 1000;      // Cases em ordem de seqüência: Case inicial
CaseLate1      = 1010;
CaseLate2      = 1020;
CaseAdv2_b     = 1100;

//.....continua
CaseIntro      = 99910;
CaseCrashed    = 99990;
CaseWaitForUserKey1 = 99991;    // hanging around waiting ctrl-[
CaseWaitForUserKey2 = 99992;    // hanging around waiting ctrl-[
CaseMovinOn    = 99993;
CaseTimeOut    = 99995;
CaseInit       = 99996;        // hang around for 10 seconds
CaseDoOnce     = 99997;        // once and only once, after init, but before lesson
CaseWait       = 99998;        // hanging around waiting for the end...
CaseEnd        = 99999;        // finished, done, kaput.
//.....continua

///
//                                     Mensagens utilizadas nessa aventura                                     //
///

//Mensagens que podem ser usadas - e concatenadas - nessa aventura.

sIdle          = "Please set the joystick throttle to idle.";
s1             = "Flight Simulator will control the aircraft while you listen to the examiner's instructions.";
s11            = "You are flying: Take off and climb to 2,000 feet.";
//.....continua

///
//                                     Declaração das Variáveis                                     //
///

```

**Esses são os casos específicos para essa aventura. Nas suas, crie seus casos de acordo com a necessidade**

**Mensagens usadas nas tolerâncias. Edite aqui também as mensagens que você vai utilizar**

**Declaração das variáveis. Todas as variáveis podem ser definidas por você mesmo mas utilizar algumas prontas economiza tempo (principal, etc.). As de checagem de tolerância podem ser todas excluídas, caso não vá utilizá-las.**

```

Var

// Variáveis exclusivas para essa lição
static number nGSDev;
//.....continua

// Variáveis de Tempo
Static Number nElapsedTime; // Tempo transcorrido da simulação desde o início
//.....continua

// Principal
Static World wTheWorld; // O Mundo (virtual da simulação)
Static AutoPilot AP; // O Piloto Automático
Static Aircraft acMe; // A Aeronave
Static Cockpit cpOffice; // Onde eu me sento
Static Shell S; // A espinha dorsal do jogo
Static Flight fltThisFlight; // O objeto de voo atual

// Outras coisas
static Number msgDisplayTime;
Static Number bFinished; // Lição encerrada, termine
//.....continua

// Variáveis das Checagens de Tolerâncias
Static Number bToleranceCheck; //Mecanismo para ligar/desligar as Checagens de Tolerâncias
//.....continua
static number nMoveOnTo;

// Variáveis de estabilização da aeronave
Number nPitch; // Nossa aeronave - pitch angle
//.....continua

// Variáveis utilizadas nas funções de mensagens
static string[3] strMsg;
static string[3] strMsgLast;
static number nMsgLevel;

```



**Vamos ver agora a seção mais extensa do script. São funções bastante úteis que você pode utilizar em suas aventuras. Todas essas funções seguem o padrão Microsoft de programação de aventuras, que você pode adotar como padrão.**

```
////////////////////////////////////
//                               //
//                               //
////////////////////////////////////
```

```
////////// Funções de Mensagens //////////
```

```
// 0) Adiciona o comentário ao resultado final da impressão na janela tipo-DOS
```

```
Function Writeln (String strString);
var
    String strLocal;
code
    strLocal = " \n";
    concat (strString,strLocal);
    Print (strString);
Endfunction;
```

**No script:...**

```
case CaseCruise1:
    Writeln("CaseCruise1");
    strAString = fn1Str1Num("Bearing =" ,nev);
    if (nDistance2Thresh < 8) then ...
```

```
// 1) básica
```

```
function fnMsg(number l, string s);           //Constrói a STRING
var
code
    if (l == Normal) then                     //simplesmente faça uma reposição para mensagens Normal
        strMsg[l] = s;
    else                                       //concatena Warnings e Errors
        if (strMsg[l] == " ") then
            strMsg[l] = s;
        else
            concat(strMsg[l],s);
        endif;
    endif;
//writeln(fn3StrBuild(fn1Str1Num("concat=====",l),": ",s))
endfunction;
```

**No script:...**

```
if (nToleranceTimer > nToleranceTimeWarn) then
    fnMsg(Error,sWarning);
    if (not bRepeated) then ...
```

```
// 2) Mostra uma mensagem na tela chamada ao término do loop do fim de cada lição
```

```
function fnMsgDisplay;
var
    number i;
    number l;
    string str;
code
    i = 0;
    l = 0;
    for i = 0 to 2 do
        l = (2 - i);
        if (strMsg[l] <> " ") then
            if (strMsg[l] <> strMsgLast[l]) then
                strMsgLast[l] = strMsg[l];
                if (nMsgLevel <= l) then
                    S.TextMessage(strMsg[l],l);
                    nMsgLevel = l;
                endif;
            endif;
            strMsg[l] = " ";
        else
            endif;
        endfor;
    endfunction;
```

**No script:...**

```
endswitch;
fnMsgDisplay;
endmodule.
```

```
// 3) Retorna a última mensagem exibida (Normal/Warning/Error)
```

```
function fnMsgLast(number l):string;
var
code
    return(strMsgLast[l]);
endfunction;
```

```
// 4) Retorna a mensagem atual exibida (Normal/Warning/Error)
```

```
function fnMsgCurrent():string;
var
code
    return(strMsgLast[nMsgLevel]);
endfunction;
```

// 4) **Sobrepõe uma mensagem (Normal/Warning/Error) com a última Normal**

```
function fnMsgRestoreNormal;
var
code
writeln("fnMsg-----RESTORE!");
    if (nMsgLevel > 0) then
        if (strMsg[0] == " ") then
            strMsg[0] = strMsgLast[0];
        endif;
        strMsg[1] = " ";
        strMsg[2] = " ";
        strMsgLast[0] = " "; //tem de eliminar a última mensagem normal para ela ser re-exibida
        strMsgLast[1] = " "; //enquanto estamos nesta, temos de eliminar a última mensagem
        strMsgLast[2] = " "; // " com mensagem de erro
        nMsgLevel = 0;
    endif;
endfunction;
```

**No script:...**

```
Writeln("turning off message");
fnMsgRestoreNormal;
nSQMessageTime = 0; ...
```

//restabelece se estamos atualmente nos modos warning/error  
//restabelece somente se não há mensagens pendentes

// 5) **Esta é mantida como cópia de compatibilidade, mas provavelmente será removida**

```
Function      fnClearMsg;
var
Code
                S.TextMessage ("",0);
                Writeln ("Clear Message");

EndFunction;
```

**No script:...**

```
case CasePostFlight:
writeln("CasePostFlight");
fnClearMsg; ...
```

////////// **Funções de STRINGS** //////////

// 1) **Constrói uma STRING baseada num número de argumentos fnStringBuildx onde x = número de argumentos**

```
Function fn4StrBuild (String One,String Two,String Three,String Four):String;
var
    String strTemp;
code
    concat (strTemp,One);
    concat (strTemp,Two);
    concat (strTemp,Three);
    concat (strTemp,Four);
    return (strTemp);
EndFunction;

Function fn3StrBuild (String One, String Two, String Three):String;
var
    String strTemp;
code
    concat (strTemp,One);
    concat (strTemp,Two);
    concat (strTemp,Three);
    return (strTemp);
EndFunction;

Function fn1Str1Num (String One, Number Two):String;
var
    String strTemp;
code
    concat (strTemp,One);
    concat (strTemp,Two);
    return (strTemp);
EndFunction;

Function fn2StrBuild (String One, String Two):String;
var
    String strTemp;
code
    concat (strTemp,One);
    concat (strTemp,Two);
    return (strTemp);
EndFunction;
```

**No script:...**

```
S.PlaySound (strwav,FALSE,0);
Writeln(fn4StrBuild("Interrupted: Now playing: ",strFileSpec," resuming ",strwav));
endif; ...
```

////////// **Funções Matemáticas** //////////

// **1) Converte metros para pés. Precisa do valor de Mtr2Ft definido.**

```
Function Meters2Feet(number nMeters):number;
Code
    return (nMeters * Mtrs2Ft);
EndFunction;
```

// **2) Converte pés para metros. Precisa do valor de Ft2Mtrs definido**

```
Function Feet2Meters(number nFeet):number;
Code
    return (nFeet * Ft2Mtrs);
EndFunction;
```

// **3) Mostra a diferença entre valores**

```
Function Diff (number a, number b):number;
var
    Number x;
code
    if (a == b) then
        x = 0;
    endif;
    if (a > b) then
        x = a - b;
    else
        x = b - a;
    endif;
    return (x);
EndFunction;
```

// **4) Acha a magnitude 'abs' de um número.**

```
Function ABS (Number nArg):Number;
code
    if (nArg >= 0) then
        return (nArg);
    else
        return (0 - nArg);
    endif;
Endfunction;
```

// **5) Function Exceeds: Checa se um argumento excedeu o limite permitido. Checa se o argumento 1 está dentro dos limites de argumento 2 e argumento 3. Espera que o argumento 2 seja o limite inferior e o argumento 3 o superior. Lembre-se que -5 é menor do que -3! Retorna um numérico FALSE se arg3 < arg1 > arg2, TRUE, o contrário.**

```
Function Exceeds (Number nVar, Number nLower, Number nUpper): Number;
var
    Number bLocalBool;
code
    bLocalBool = FALSE;
    if (nVar < nLower) then
        bLocalBool = TRUE;
    endif;
    if (nVar > nUpper) then
        bLocalBool = TRUE;
    endif;
    return (bLocalBool);
EndFunction;
```

// Este será o valor retornado  
// Espere pelo melhor  
// muito pequeno?  
// Sim.  
// muito pequeno?  
// Sim.  
// Retorna o valor true/false para o chamador.

// **6) Diff360 - Esta rotina computa a diferença entre as direções. Argumento: Heading1, Heading2: números; Retorna: A diferença (registrada) entre as duas**

```
Function Diff360 (Number Hdg1, Number Hdg2):Number;
Code
    if (Diff (Hdg1,Hdg2) <= 180) then
        return (Hdg1 - Hdg2);
    endif;
    if ( ( Hdg1 > 180) and (Hdg2 < 180) ) then
        Hdg2 = Hdg2 + 360;
        return (Hdg1 - Hdg2);
    endif;
    if ( ( Hdg1 < 180) and (Hdg2 > 180) ) then
        Hdg1 = Hdg1 + 360;
        return (Hdg1 - Hdg2);
    endif;
    strAStrng = "Well, we're here! Hdg1 = ";
    concat (strAStrng,Hdg1);
    concat (strAStrng,", Hdg2 = ");
    concat (strAStrng,Hdg2);
    Writeln (strAStrng);
EndFunction;
```

// caso 1: o mais fácil  
// caso 2: isto é 300 vs 010  
// Ajuste o mais para leste  
// caso 3: isto é 010 vs 300  
// novamente, ajuste o mais para leste  
// Nós não deveríamos nunca estar aqui.

**No script:...**

```
if (Diff(nYaw,(kHdg1 + 30 + nTurnRate + 5)) < 10) and (bPassGo) then
    bRollCheck = FALSE;
    fnSQ_NormalMsg(s13b,0);
```

**// 7) Normalize360 - Esta rotina coloca a direção da proa entre 0 e 360**

**Argumentos: Hdg: número**

**Retorna: Uma direção equivalente entre 0 e 360**

**Efeitos colaterais: Nenhum (do que eu sei)**

```
Function Normalize360 (Number Hdg):Number;
code
    while (Hdg>360) do Hdg=Hdg-360; endwhile;
    while (Hdg<0) do Hdg=Hdg+360; endwhile;
    return (Hdg);
EndFunction;
```

**// 8) Retorna o máximo dos dois valores**

```
Function Max (number a, number b):number;
var
    Number x;
code
    if (a == b) then x = a; endif;
    if (a > b) then x = a; endif;
    if (a < b) then x = b; endif;
    return (x);
EndFunction;
```

**// 9) Retorna o mínimo dos dois valores**

```
Function Min (number a, number b):number;
var
    Number x;
code
    if (a == b) then x = a; endif;
    if (a < b) then x = a; endif;
    if (a > b) then x = b; endif;
    return (x);
EndFunction;
```

**////////// Funções de Aceleração da Aeronave //////////**

**// 1) Esta rotina ajusta a aceleração para um valor específico de um período de tempo e vai repetir o valor correto.**

**// Esta é uma função \*PARTICULAR\* - NÃO CHAME-A \*\* Use fnThrottle\_Hold, ao invés dessa.**

```
function fnThrottle_SetEngines(Number Throttle, Number Mixture, Number prop);
var
    Number x;
code
    for x = 0 to (nEngineCount-1) do
        ecEngines[x].Throttle = Throttle;
        ecEngines[x].Mixture = Mixture;
        ecEngines[x].Prop = Prop;
    endfor;
endfunction;
```

**// 2) Essa rotina ajusta a aceleração a uma RPM específica, e deve ser chamada diversas vezes por um período de tempo e depois vai repetir o valor correto. Esta é uma função \*PARTICULAR\* - NÃO CHAME-A \*\* Use fnThrottle\_HoldRPM, ao invés dessa**

```
function fnThrottle_SetToRPM(Number RPM);
var
    Number nTach; // leitura atual do tacômetro
    Number nThrot; // valor atual da aceleração
    Number deltaThrot;
code
    nTach = eMotors[0].RPM; // get the current RPM
    nThrot = ecEngines[0].Throttle;
    if (diff (nTach,RPM) > 50) then
        deltaThrot = 0.05;
        if (diff(nTach,RPM)>250) then deltaThrot = 0.1; endif;
        if (diff(nTach,RPM)>500) then deltaThrot = 0.25; endif;
        if (RPM>nTach) then
            nThrot = nThrot + deltaThrot;
        else
            nThrot = nThrot - deltaThrot;
        endif;
        if (nThrot < 0) then nThrot = 0; endif;
        if (nThrot > 1) then nThrot = 1; endif;
        if (Diff (nTach,nTachPrev) < 50) then
            if (RPM > nTach) then
                fnThrottle_SetEngines((nThrot),1.0,1.0);
            else
                fnThrottle_SetEngines((nThrot),1.0,1.0);
            endif;
        else
            if ( (nTach>RPM) AND (nTach>nTachPrev) ) then
                fnThrottle_SetEngines((nThrot),1.0,1.0);
            endif;
            if ( (nTach<RPM) AND (nTach<nTachPrev) ) then
                fnThrottle_SetEngines((nThrot),1.0,1.0);
            endif;
        endif;
        nTachPrev = nTach;
endfunction;
```



*// 3) Essa rotina controla a aceleração. Ela verifica se alguma coisa precisa ser modificada e conduz de forma correta. Essa rotina nunca deve ser chamada de scripts "normais" - deve ser chamada uma vez no início de cada loop.*

```
function fnThrottle_Manage;
    var
    code
        switch (nThrottleHold)
            case THROTTLE_MANUAL: endcase;
            case THROTTLE_LEVEL: fnThrottle_SetEngines(nThrottleValue,1.0,1.0); endcase;
            case THROTTLE_RPM:   fnThrottle_SetToRPM(nThrottleValue); endcase;
        endswitch;
endfunction;
```

*// 4) Essa rotina fixa o nível de aceleração a uma porcentagem fixa*

```
function fnThrottle_Hold(Number nThrottle);
    code
        nThrottleValue = nThrottle;
        nThrottleHold = THROTTLE_LEVEL;
        fnThrottle_Manage;
endfunction;
```

*// 5) Essa rotina fixa o nível de aceleração a uma RPM fixa*

```
function fnThrottle_HoldRPM(Number nThrottle);
    code
        nThrottleValue = nThrottle;
        nThrottleHold = THROTTLE_RPM;
        fnThrottle_Manage;
endfunction;
```

*// 6) Essa rotina libera a aceleração para controle manual*

```
function fnThrottle_Manual;
    code
        nThrottleHold = THROTTLE_MANUAL;
        fnThrottle_Manage;
endfunction;
```

*// 7) Essa rotina desliga os motores*

```
function fnThrottle_Shutdown;
    code
        nThrottleHold = THROTTLE_MANUAL;
        fnThrottle_SetEngines(0,0,0);
endfunction;
```

*////////// Funções de Encadeamento de Mensagens de Texto e Áudio //////////*

*// Esta rotina muda o estado (state). Esta Rotina é privada - NÃO A CHAME DIRETAMENTE DAQUI*

```
function fnSQ_PrivateChangeState;
    var
    code
        String str;
        nState = nSQState[nSQCurrent];
        nSQQuiet = nSQTime[nSQCurrent];
        str = fn2StrBuild(fn1Str1Num("SQ State: ",nState),fn1Str1Num("Quiet: ",nSQQuiet));
        Writeln(str);
endfunction;
```

*// Esta rotina PAUSA. Esta Rotina é privada - NÃO A CHAME DIRETAMENTE DAQUI*

```
function fnSQ_PrivatePause;
    var
    code
        String str;
        nSQQuiet = nSQTime[nSQCurrent];
        str = fn1Str1Num("SQ Pause: ",nSQQuiet);
        Writeln(str);
endfunction;
```

*// Esta rotina TOCA UM ARQUIVO DE SOM \*.WAV (WAVE). Rotina privada - NÃO A CHAME DIRETAMENTE DAQUI*

```
function fnSQ_PrivatePlay;
    var
    code
        String strWav;
        String str;
        strWav = strSQStrings[nSQCurrent];
        str = fn2StrBuild("Playing: ",strWav);
        S.PlaySound (strWav,FALSE,0); // Toca o som
        Writeln(str);
endfunction;
```

// **Esta rotina EXIBE UMA MENSGAEM DE TEXTO. Rotina privada - NÃO A CHAME DIRETAMENTE DAQUI**

```
function fnSQ_PrivateMsg;
var
    String strMsg;
    String str;
code
    strMsg = strSQStrings[nSQCurrent];
    if (nSQTime[nSQCurrent] == 0) then
        nSQMessageTime= 0; //não apague essa mensagem automaticamente
    else
        nSQMessageTime= nElapsedTime + nSQTime[nSQCurrent];
    endif;
    fnMsg(nSQState[nSQCurrent],strMsg);
    str = fn2StrBuild("SQ Message: ",strMsg);
    concat(str," type: ");
    concat(str,nSQState[nSQCurrent]);
    concat(str," off: ");
    concat(str,nSQMessageTime);
    Writeln(str);
endfunction;
```

// **Esta rotina REALÇA UM MOSTRADOR. Rotina privada - NÃO A CHAME DIRETAMENTE DAQUI**

```
function fnSQ_PrivateGaugeHilight;
var
    String str;
code
    if (nSQGaugeTime > 0) then
        acMe.GaugeDisplay(strSQCurrentGauge,"Normal");
    endif;
    strSQCurrentGauge = strSQStrings[nSQCurrent];
    nSQGaugeTime= nElapsedTime + nSQTime[nSQCurrent];
    acMe.GaugeDisplay(strSQCurrentGauge ,"Hilight");
    str = fn2StrBuild("Hilight: ",strSQCurrentGauge );
    concat(str," off: ");
    concat(str,nSQGaugeTime);
    Writeln(str);
endfunction;
```

// **Esta rotina INICIA UM COMEÇO RETARDADO DA CHECAGEM DE TOLERÂNCIA DOS MOTORES. Rotina privada**

```
function fnSQ_PrivateTolerance;
var
    String str;
code
    bToleranceCheck = TRUE;
    sWarningLast = " ";
    fnMsgRestoreNormal;
    nToleranceStartTime = (nElapsedTime + nSQTime[nSQCurrent]);
    bWithinTolerance = FALSE;
    writeln(fn1str1num("SQ Delayed Tolerance Engine Start: ",nToleranceStartTime));
endfunction;
```

// **Esta rotina AJUSTA O INÍCIO RETARDADO DE UMA TOLERÂNCIA INDIVIDUAL EM MOVIMENTO. Rotina privada**

```
function fnSQ_PrivateToleranceFlag;
var
    string str;
code
    //Simplesmente acerte o tempo de início. Os ajustes reais serão feitos automaticamente.
    nTFStart[nSQState[nSQCurrent]] = (nElapsedTime + nSQTime[nSQCurrent]);
    concat(str,"Delay Tolerance #");
    concat(str,nSQState[nSQCurrent]);
    concat(str," for ");
    concat(str,nSQTime[nSQCurrent]);
    concat(str," seconds: ");
    concat(str,(nElapsedTime + nSQTime[nSQCurrent]));
    writeln(str);
endfunction;
```

// **Esta rotina REINICIA O TEMPO (CRONOMETRAGEM DA SIMULAÇÃO). Rotina Privada.**

```
function fnSQ_PrivateResetTimer;
var
code
    // limpa qualquer sujeira
    nTimer = 0;
    Writeln("SQ Timer: reset");
endfunction;
```

// **Esta rotina AJUSTA A ACELERAÇÃO. Rotina Privada**

```
function fnSQ_PrivateThrottle;
var
    String str;
code
    switch nSQState[nSQCurrent]
        case THROTTLE_MANUAL:
            fnThrottle_Manual;
            str = "SQ Throttle: Manual";
        endcase;
        case THROTTLE_LEVEL:
            fnThrottle_Hold(nSQTime[nSQCurrent]);
            str = fn1Str1Num("SQ Throttle: Hold ",nSQTime[nSQCurrent]);
        endcase;
        case THROTTLE_RPM:
            fnThrottle_HoldRPM(nSQTime[nSQCurrent]);
            str = fn1Str1Num("SQ Throttle: HoldRPM ",nSQTime[nSQCurrent]);
        endcase;
    endswitch;
    Writeln(str);
endfunction;
```

// **Esta rotina ADICIONA UMA INSTRUÇÃO NA FILA. Rotina Privada**

```
function fnSQ_PrivateAppend(Number nType,String strString,Number n_Time,Number n_State):Number;
var
    Number next;
code
    next = nSQNext+1;
    if ( next >= nSQLength ) then next = next - nSQLength; endif;
    if ( next == nSQCurrent) then
        Writeln ("Error: sound queue is full"); //avisando que devemos fazer algo
        return (0);
    endif;
    strSQStrings[nSQNext] = strString;
    nSQState[nSQNext] = n_State;
    nSQTime[nSQNext] = n_Time;
    nSQType[nSQNext] = nType; // event type
    return (next);
endfunction;
```

// **Esta rotina GERENCIA A FILA. Rotina Privada**

```
function fnSQ_Private(Number nType,String strString,Number n_Time,Number n_State,Number nInsert);
var
    String strMsg;
    Number next;
code
    if (nInsert) then
        next = fnSQ_PrivatePrepend(nType,strString,n_Time,n_State);
    else
        next = fnSQ_PrivateAppend(nType,strString,n_Time,n_State);
    endif;
    strMsg = fn1Str1Num("Queued: ",nType);
    concat(strMsg," time: ");
    concat(strMsg,n_Time);
    concat(strMsg," state: ");
    concat(strMsg,n_State);
    concat(strMsg," str: ");
    concat(strMsg,strString);
    if (nSQCurrent == nSQNext) then // toca o som se nada estiver sendo tocado
        switch(nType)
            case SQ_PLAY: fnSQ_PrivatePlay; endcase;
            case SQ_PAUSE: fnSQ_PrivatePause; endcase;
            case SQ_STATE: fnSQ_PrivateChangeState; endcase;
            case SQ_MESSAGE: fnSQ_PrivateMsg; endcase;
            case SQ_GAUGEHLIGHT: fnSQ_PrivateGaugeHiligh; endcase;
            case SQ_THROTTLE: fnSQ_PrivateThrottle; endcase;
            case SQ_RESETTIMER: fnSQ_PrivateResetTimer; endcase;
            case SQ_TOLERANCE: fnSQ_PrivateTolerance; endcase;
            case SQ_TOLERANCEFLAG: fnSQ_PrivateToleranceFlag; endcase;
        endswitch;
    endif;
    nSQNext = next;
    Writeln(strMsg);
endfunction;
```

**// Esta rotina GERENCIA A FILA DE SONS. ELA VERIFICA SE ALGO ESTÁ SENDO TOCADO E SE ALGO ESTÁ NA ESPERA DE SER TOCADO E EXECUTA A AÇÃO DE ACORDO. ESTA ROTINA NUNCA DEVE SER CHAMADA POR SCRIPTS NORMAIS - DEVE SER CHAMADA UMA VEZ NO INÍCIO DE CADA LOOP. Esta Rotina é a última das rotinas privadas.**

```

function fnSQ_Manage;
var
    String str;   String strwav;   Number length;   Number etype;
code
    // Toca o próximo som se nada está atualmente sendo tocado e se há algo na fila.
    nSQRemaining = S.GetRemainingSoundTime;
    str = "SQ remaining: ";
    concat(str,nSQRemaining );
    concat(str," quiet: ");
    concat(str,nSQQuiet);
    concat(str," state: ");
    concat(str,nState);
    concat(str," cur: ");
    concat(str,nSQCurrent);
    concat(str," next: ");
    concat(str,nSQNext);
    if ( (nSQRemaining > 0) AND (bUserKey2) ) then
        Writeln("userkey2 advances soundqueue");
        bUserKey2 = FALSE;
        S.AdvanceSoundQueue; // pop the queue
        S.PlaySound ("",FALSE,0); // Clear the wave & text
        nSQRemaining = 0;
        nSQQuiet = 0;
    endif;
    if (nState==CaseWaitForUserKey1) then
        if (bUserKey1) then
            nSQQuiet = 0;
            bUserKey1 = FALSE; // consumir o evento
        else
            nSQQuiet = 1; // força uma pausa
        endif;
    endif;
    if (nState==CaseWaitForUserKey2) then
        if (bUserKey2) then
            nSQQuiet = 0;
            bUserKey2 = FALSE; // consumir o evento
        else
            nSQQuiet = 1; // força uma pausa
        endif;
    endif;
    if ( (nSQMessageTime>0) AND (nElapsedTime>nSQMessageTime) ) then // desliga qualquer mensagem
        Writeln("turning off message");
        fnMsgRestoreNormal;
        nSQMessageTime = 0;
    endif;
    // Limpa todos os mosradores
    if ( (nSQGaugeTime>0) AND (nElapsedTime>nSQGaugeTime) ) then
        acMe.GaugeDisplay(strSQCurrentGauge,"Normal");
        nSQGaugeTime = 0;
    endif;
    if ( nSQRemaining <= 0 ) then // if nothing is playing
        if (nSQQuiet <=0) then // if e're not pausing
            while ( (nSQRemaining <= 0) AND (nSQQuiet <= 0) AND (nSQCurrent <> nSQNext) ) do
                // loop through queue contents until we are busy
                nSQCurrent = nSQCurrent + 1;
            if( nSQCurrent >= nSQLength ) then nSQCurrent = nSQCurrent - nSQLength; endif;
            if( nSQCurrent <> nSQNext ) then
                etype = nSQType[nSQCurrent];
                switch(etype)
                    case SQ_PLAY:
                        fnSQ_PrivatePlay; nSQRemaining = S.GetRemainingSoundTime; endcase;
                    case SQ_PAUSE:
                        fnSQ_PrivatePause; endcase;
                    case SQ_STATE:
                        fnSQ_PrivateChangeState; endcase;
                    case SQ_MESSAGE:
                        fnSQ_PrivateMsg; endcase;
                    case SQ_GAUGEHLIGHT:
                        fnSQ_PrivateGaugeHilight; endcase;
                    case SQ_THROTTLE:
                        fnSQ_PrivateThrottle; endcase;
                    case SQ_RESETTIMER:
                        fnSQ_PrivateResetTimer; endcase;
                    case SQ_TOLERANCE:
                        fnSQ_PrivateTolerance; endcase;
                    case SQ_TOLERANCEFLAG:
                        fnSQ_PrivateToleranceFlag; endcase;
                endswitch;
            endif;
        endwhile;
    else
        nSQQuiet = nSQQuiet - nDeltaTime;
    endif;
endfunction;

```



// **Esta rotina LIMPA TODOS OS SONS E LEGENDAS**

```
function fnSQ_ClearAll;
var
code
    S.AdvanceSoundQueue;           // dispara a fila
    S.PlaySound ("",FALSE,0);      // Limpa o som e o texto
    nQuietTime = 0;
    nSQCurrent = 0;
    nSQNext = 0;
    Writeln ("SoundQueue.ClearAll");
endfunction;
```

// **Esta rotina SOMENTE TOCA UM ARQUIVO WAVE (DE SOM com ou sem legendas) - NÃO HÁ MUDANÇA DE ESTADO (STATE)**

```
function fnSQ_Play(String Filename);
var
    String strFileSpec;
code
    strFileSpec = fn3StrBuild(strWavBase,Filename,".WAV"); // create the filespec
    fnSQ_Private(SQ_Play,strFileSpec,0,0,FALSE);
endfunction;
```

// **Esta rotina INSERE UM ARQUIVO WAVE - NÃO HÁ MUDANÇA DE ESTADO (STATE)**

```
function fnSQ_Insert(String Filename);
var
    String strFileSpec;
code
    strFileSpec = fn3StrBuild(strWavBase,Filename,".WAV"); // create the filespec
    fnSQ_Private(SQ_Play,strFileSpec,0,0,TRUE);
endfunction;
```

// **Esta rotina muda o estado de um estado quando o outro terminou**

```
function fnSQ_ChangeState(Number nNewState);
code
    switch(nNewState)
        case 0: return; endcase;
        case CaseWaitForUserKey1:
            fnSQ_Private(SQ_STATE,"",1,nNewState,FALSE); // minimum 1 second/1 loop pause
            return;
        endcase;
        case CaseWaitForUserKey2:
            fnSQ_Private(SQ_STATE,"",1,nNewState,FALSE); // minimum 1 second/1 loop pause
            return;
        endcase;
    endswitch;
    fnSQ_Private(SQ_STATE,"",0,nNewState,FALSE);
endfunction;
```

// **Esta rotina insere algum silêncio após um arquivo wave ter sido tocado**

```
function fnSQ_Pause(Number nPause);
code
    fnSQ_Private(SQ_PAUSE,"",nPause,0,FALSE);
endfunction;
```

// **Esta rotina toca um arquivo wave interrompendo o atual, continuando com a programação agendada**

```
function fnSQ_Interrupt(String Filename);
var
    String strFileSpec;
    String strwav;
code
    strFileSpec = fn3StrBuild(strWavBase,Filename,".WAV"); // create the filespec
    // toca o próximo som se nada está sendo tocado e se tem algum na fila
    Writeln(fn1Str1Num("SQ_Interrupt remaining: ",nSQRemaining));
    if (nSQRemaining<=0) then
        // não há nada ainda tocando, somente toque isso
        S.PlaySound (strFileSpec,FALSE,0);
        Writeln(fn2StrBuild("nothing to interrupt: ",strFileSpec));
    else
        S.AdvanceSoundQueue;
        S.PlaySound ("",FALSE,0);
        S.PlaySound (strFileSpec,FALSE,0);
        strwav = strSQStrings[nSQCurrent];
        S.PlaySound (strwav,FALSE,0);
        Writeln(fn4StrBuild("Interrupted: Now playing: ",strFileSpec," resuming ",strwav));
    endif;
endfunction;
```

```

// Esta rotina exibe uma mensagem de texto NORMAL por um determinado período de tempo
function fnSQ_NormalMsg(String strMsg,Number nPause);
code
    fnSQ_Private(SQ_MESSAGE,strMsg,nPause,0,FALSE);
endfunction;

// Esta rotina exibe uma mensagem de texto WARNING por um determinado período de tempo
function fnSQ_WarningMsg(String strMsg,Number nPause);
code
    fnSQ_Private(SQ_MESSAGE,strMsg,nPause,1,FALSE);
endfunction;

// Esta rotina exibe uma mensagem de texto ERROR por um determinado período de tempo
function fnSQ_ErrorMsg(String strMsg,Number nPause);
code
    fnSQ_Private(SQ_MESSAGE,strMsg,nPause,2,FALSE);
endfunction;

// Esta rotina realça um mostrador por um período de tempo
function fnSQ_GaugeHilight(String strGauge,Number nPause);
code
    fnSQ_Private(SQ_GAUGEHLIGHT,strGauge,nPause,0,FALSE);
endfunction;

// Esta rotina reinicia o cronômetro em zero
function fnSQ_ResetTimer;
code
    fnSQ_Private(SQ_RESETTIMER,"",0,0,FALSE);
endfunction;

// Esta rotina ativa o mecanismo de tolerância
function fnSQ_Tolerance(Number nPause);
code
    fnSQ_Private(SQ_TOLERANCE,"",nPause,0,FALSE);
endfunction;

// Esta ativa uma tolerância individual
function fnSQ_ToleranceFlag(Number nFlag, Number nPause);
code
    fnSQ_Private(SQ_TOLERANCEFLAG,"",nPause,nFlag,FALSE);
endfunction;

// Esta ajusta a aceleração. Idêntica às anteriores
function fnSQ_ThrottleManual;
code
    fnSQ_Private(SQ_THROTTLE,"",0,THROTTLE_MANUAL,FALSE);
endfunction;

function fnSQ_ThrottleHold(Number nPercent);
code
    fnSQ_Private(SQ_THROTTLE,"",nPercent,THROTTLE_LEVEL,FALSE);
endfunction;

function fnSQ_ThrottleHoldRPM(Number nRPM);
code
    fnSQ_Private(SQ_THROTTLE,"",nRPM,THROTTLE_RPM,FALSE);
endfunction;

// Esta rotina retorna FALSE se um som está sendo tocado ou agendado. Ou TRUE se o agendamento está vazio
function fnSQ_isEmpty:Number;
var
    String str;
code
    str = "isEmpty remaining: ";
    concat(str,nSQRemaining);
    concat(str," cur: ");
    concat(str,nSQCurrent);
    concat(str," next: ");
    concat(str,nSQNext);
    // Writeln(str);
    if ((nSQRemaining < 0) AND (nSQCurrent == nSQNext)) then
        return(TRUE);
    endif;
    return(FALSE);
endfunction;

```

# ////////// Funções Assíncronas //////////

*// 1) Manipula os eventos assíncronos de entrada usando onUserEvent1 & onUserEvent2. Define as variáveis globais bUserKey1 (CTRL-[) ou bUserKey2 (CTRL-]) apropriadamente.*

```
function onUserEvent1;
code
    bUserKey1 = TRUE;
    Writeln ("Key 1");
endfunction;

function onUserEvent2;
code
    bUserKey2 = TRUE;
    Writeln ("Key 2");
endfunction;

function on_Crash;
code
    nState = CaseCrashed;
endfunction;

function resetUserEvents;
code
    bUserKey1 = FALSE;
    bUserKey2 = FALSE;
endfunction;
```

# ////////// Funções de Critérios de Tolerâncias //////////

*// 1) Reseta os avisos de tolerância*

```
function fnResetToleranceFlags;
var
code
    bWithinTolerance = FALSE;
    //.....continua
```

*// 2) Ativa/Desativa a checagem das tolerâncias*

```
function fnToleranceChecking(number bOnOff);
var
code
    bToleranceCheck = bOnOff;
    sWarningLast = " ";
    fnMsgRestoreNormal;
    Writeln(fn1Str1Num("Tolerance Checking: ",bToleranceCheck));
endfunction;
```

*// 3) Público - Ativa ou inativa checagem de tolerâncias(nota: o delay (espera) funciona para retardar o início.*

```
function fnTolerance(number nFlag, number bOnOff, number nDelay);
var
    string str;
code
    if (nDelay == 0) then
        switch nFlag
        //.....continua
```

*// 4) Liga os avisos de checagem retardados*

```
function fnToleranceStart();
var
    number nCounter;
code
    for nCounter = 0 to 24 do
        if (nTFStart[nCounter] <> 0) then //non 0 is an active timer
            if ((nTFStart[nCounter] - nElapsedTime) <= 0) then
                fnTolerance(nCounter,ON,0);
            endif;
        endif;
    endfor;
endfunction;
```

*// 5) Início retardado das Tolerâncias dos Motores. Esta função é repostada pela fnSQ\_ToleranceFlag. Ela só está ainda aqui até que o pessoal da Microsoft atualize as lições que ainda a utilizem*

```
function fnToleranceStartDelay(number nSeconds);
var
code
    nToleranceStartTime = (nElapsedTime + nSeconds);
    bWithinTolerance = FALSE;
    fnToleranceChecking(TRUE);
    writeln(fn1str1num("Delayed Tolerance Start: ",nToleranceStartTime));
endfunction;
```

**// 6) Avaliam todas as tolerâncias. \* Privada \* Não chame no geral**

```
function fnCheckTolerances;
```

```
    //.....continua
```

**// 7) Esta rotina não é realmente uma tolerância - O que está fazendo aqui? retorna TRUE se a aeronave não está se movendo**

```
function IsStopped():number;
```

```
var
```

```
    Number bStopped;
```

```
code
```

```
    if ((Diff(posHere.Latitude, nLastLat ) < 0.000001)
        and (Diff(posHere.Longitude,nLastLon ) < 0.000001) ) then
        bStopped = TRUE;
```

```
    else
```

```
        bStopped = FALSE;
```

```
    endif;
```

```
    nLastLat = posHere.Latitude;
```

```
    nLastLon = posHere.Longitude;
```

```
    return (bStopped);
```

```
endfunction;
```

**// 8) Retorna valor dentro das altitudes limites**

```
function fnAtAltitude():number;
```

```
var
```

```
code
```

```
    return ((nAltitude >= (nAltitudeTarget + nAltitudeToleranceMin))
        and (nAltitude <= (nAltitudeTarget + nAltitudeToleranceMax)))
```

```
endfunction;
```

**// 9) Retorna Cowl Flaps**

```
function fnAtCowlFlaps():number;
```

```
var
```

```
    number bCowlFlaps;
```

```
    number nCounter;
```

```
code
```

```
    bCowlFlaps = TRUE;
```

```
    for nCounter = 0 to (acMe.NumEngines - 1) do
```

```
        bCowlFlaps = bCowlFlaps and ((ecEngines[nCounter].CowlFlaps - nCowlFlapsTarget) < nCowlFlapsToleranceMax)
```

```
        and ((ecEngines[nCounter].CowlFlaps - nCowlFlapsTarget) > nCowlFlapsToleranceMin);
```

```
        writeln(fn1str1num("CF :",ecEngines[nCounter].CowlFlaps));
```

```
        writeln(fn1str1num("CFx:",nCowlFlapsTarget));
```

```
        writeln(fn1str1num("CF+:",nCowlFlapsToleranceMax));
```

```
        writeln(fn1str1num("CF-:",nCowlFlapsToleranceMin));
```

```
    endfor;
```

```
    return (bCowlFlaps)
```

```
endfunction;
```

**// 10) Retorna posição dos Flaps**

```
function fnAtFlaps():number;
```

```
var
```

```
code
```

```
    return (cpOffice.FlapsHandlePos == nFlapsTarget)
```

```
endfunction;
```

**// 11) Retorna Glideslope**

```
function fnAtGlideSlope():number;
```

```
var
```

```
code
```

```
    return (abs(nGlideSlope) > nGlideSlopeTolerance)
```

```
endfunction;
```

**// 12) Retorna Curso**

```
function fnAtHeading():number;
```

```
var
```

```
code
```

```
    return ((diff360(nYaw,nHeadingTarget) >= nHeadingToleranceMin)
        and (diff360(nYaw,nHeadingTarget) <= nHeadingToleranceMax))
```

```
endfunction;
```

**// 13) Retorna Trem de Pouso**

```
function fnAtLandingGear():number;
```

```
var
```

```
code
```

```
    return (round(cpOffice.LandingGearHandlePos) == bLandingGear737Target)
```

```
endfunction;
```



**// 14) Retorna Localizador**

```
function fnAtLocalizer():number;
var
code
    return (((vorSelect.ToFrom == 1) and (abs(nLocalizer) < nLocalizerTolerance))
            or ((vorSelect.ToFrom == 2) and ((180-abs(nLocalizer)) < nLocalizerTolerance)))
endfunction;
```

**// 15) Retorna Mistura dos Motores**

```
function fnAtMP():number;
var
    number bMP;
    number nCounter;
code
    bMP = TRUE;
    for nCounter = 0 to (acMe.NumEngines - 1) do
        bMP = bMP and ((eMotors[nCounter].ManifoldPressure - nMPTarget) < nMPToleranceMax)
        and ((eMotors[nCounter].ManifoldPressure - nMPTarget) > nMPToleranceMin);
        writeln(fn1str1num("MP :",eMotors[nCounter].ManifoldPressure));
        writeln(fn1str1num("MPx:",nMPTarget));
        writeln(fn1str1num("MP+:",nMPToleranceMax));
        writeln(fn1str1num("MP-:",nMPToleranceMin));
    endfor;
    return (bMP)
endfunction;
```

**// 16) Retorna Freios de estacionamento**

```
function fnAtParkingBrake():number;
var
code
    return (cpOffice.ParkingBrakeOn == bParkingBrakeTarget)
endfunction;
```

**// 17) Retorna Pitch**

```
function fnAtPitch():number;
var
code
    return ((nPitch >= (nPitchTarget + nPitchToleranceMin))
            and (nPitch <= (nPitchTarget + nPitchToleranceMax)))
endfunction;
```

**// 18) Retorna Propulsão dos Motores**

```
function fnAtProp():number;
var
    number bProp;
    number nCounter;
code
    bProp = TRUE;
    for nCounter = 0 to (acMe.NumEngines - 1) do
        bProp = bProp and ((ecEngines[nCounter].Prop - nPropTarget) < nPropToleranceMax)
        and ((ecEngines[nCounter].Prop - nPropTarget) > nPropToleranceMin);
        writeln(fn1str1num("MP :",ecEngines[nCounter].Prop));
        writeln(fn1str1num("MPx:",nPropTarget));
        writeln(fn1str1num("MP+:",nPropToleranceMax));
        writeln(fn1str1num("MP-:",nPropToleranceMin));
    endfor;
    return (bProp)
endfunction;
```

**// 19) Retorna Rolagem**

```
function fnAtRoll():number;
var
code
    return ((nRoll >= (nRollTarget + nRollToleranceMin))
            and (nRoll <= (nRollTarget + nRollToleranceMax)))
endfunction;
```

**// 20) Retorna RPM dos Motores**

```
function fnAtRPM():number;
var
code
    return ((eMotors[0].RPM >= (nRPMTarget + nRPMToleranceMin))
            and (eMotors[0].RPM <= (nRPMTarget + nRPMToleranceMax)))
endfunction;
```

**// 21) Retorna Pistas**

```
function fnAtRunway():number;
var
    number nLat;
    number nLon;
code
    nLat = posHere.Latitude;
    nLon = posHere.Longitude;
    return ((nLat - nPoint[0,LAT] - (nPoint[1,LAT]-nPoint[0,LAT])/(nPoint[1,LON]-nPoint[0,LON])*(nLon - nPoint[0,LON]) < 0) and
        (nLat - nPoint[1,LAT] - (nPoint[2,LAT]-nPoint[1,LAT])/(nPoint[2,LON]-nPoint[1,LON])*(nLon - nPoint[1,LON]) < 0) and
        (nLat - nPoint[2,LAT] - (nPoint[3,LAT]-nPoint[2,LAT])/(nPoint[3,LON]-nPoint[2,LON])*(nLon - nPoint[2,LON]) > 0) and
        (nLat - nPoint[3,LAT] - (nPoint[0,LAT]-nPoint[3,LAT])/(nPoint[0,LON]-nPoint[3,LON])*(nLon - nPoint[3,LON]) > 0));
endfunction;
```

**// 22) Retorna Velocidade de solo**

```
function fnAtSpeed():number;
var
code
    return ((nIASNow >= (nSpeedTarget + nSpeedToleranceMin))
        and (nIASNow <= (nSpeedTarget + nSpeedToleranceMax)))
endfunction;
```

**// 23) Retorna Parado**

```
function fnAtStop():number;
var
    Number bStopped;
code
    bStopped = ((Diff(posHere.Latitude, nLastLat ) < 0.000001)
        and (Diff(posHere.Longitude,nLastLon ) < 0.000001));
    nLastLat = posHere.Latitude;
    nLastLon = posHere.Longitude;
    return(bStopped);
endfunction;
```

**// 24) Retorna Aceleração**

```
function fnAtThrottle():number;
var
code
    return ((ecEngines[0].Throttle >= (nThrottleTarget + nThrottleToleranceMin))
        and (ecEngines[0].Throttle <= (nThrottleTarget + nThrottleToleranceMax)))
endfunction;
```

**// 25) Retorna Velocidade Vertical**

```
function fnAtVerticalSpeed():number;
var
code
    return (((nVSnw - nVertSpeedTarget) >= nVertSpeedToleranceMin)
        and ((nVSnw - nVertSpeedTarget) <= nVertSpeedToleranceMax))
endfunction;
```

**// Esta rotina chama CaseWait se há uma outra mudança de caso vindo, caso contrário não faz nada.**

```
function fnSQ_Synchronize;

var
    String str;
    Number current;
    Number callit;

code
    callit = FALSE;
    current = nSQCurrent;
    while ((not callit) AND (current <> nSQNext) ) do // loop through queue contents looking for non-CaseWait events
        current = current + 1;
        if( current >= nSQLength ) then current = current - nSQLength; endif;
        if( current <> nSQNext ) then
            if ( (nSQType[current] == SQ_STATE) and (nSQState[current] <> 0) and (nSQState[current] <> CaseWait) ) then
                callit = TRUE;
            endif;
        endif;
    endwhile;
    if (callit == TRUE) then
        nState = CaseWait;
    endif;
    Writeln(fn1Str1Num("SQ Synchronize ",callit));
endfunction;
```

////////// **Função de Inicialização** //////////

```
function Init;
var
code
```

**Variáveis globais para inicializações das lições**

**// Define os objetos básicos**

```
S = getShell; // A "espinha dorsal" do FS
S.PlaySound("",0,0);
Writeln("Initializing");
wTheWorld = getWorld; // Objeto Mundo
acMe = wTheWorld.AircraftByID(0); // Objeto Aeronave
AP = acMe.Autopilot; // Objeto Piloto Automático
fltThisFlight = wTheWorld.CurrentFlight; // Objeto Vôo Atual
```

**// Define os membros dos objetos básicos**

```
cpOffice = acMe.Cockpit; // Objeto Cockpit
nEngineCount = acMe.NumEngines; // Número de motores
```

**Outros valores que precisam ser definidos durante a inicialização ABL pedem que as variáveis sejam "definidas" antes de serem "lidas" devido às suas características de linguagem**

**// Variáveis de Tempo**

```
nElapsedTime = 0;
nDeltaTime = 0;
nQuietTime = 0; // Tempo para encerrar antes de tocar próximo wave
nQuietTime2 = 0;
nTimer = 0; // Tempo Genérico
nManTime = 0;
```

**// Outras coisas**

```
nState = CaseInit; // CASE inicial para essa lição.
bWOW = TRUE; // Sim, estamos no chão
bFinished = FALSE; // Acabamos de começar...
strWavBase = "sound\\lessons\\"; // Pasta com os arquivos de som
nDoneFlag = 0;
bCrashCheck = TRUE;
bUserKey1 = FALSE;
bUserKey2 = FALSE;
```

**// Define o valor inicial das mensagens**

```
strMsg[0] = " "; //tem de usar o espaço entre aspas pro ABL reconhecer
strMsg[1] = " ";
strMsg[2] = " ";
strMsgLast[0] = " ";
strMsgLast[1] = " ";
strMsgLast[2] = " ";
nMsgLevel = 0;
```

**// Define os valores iniciais das tolerâncias**

```
for nLooper = 0 to 24 do
    nTFStart[nLooper] = 0;
endfor;
//.....continua
```

**// Ajustes Iniciais de potência do motor - desligados**

```
fnNormalMsg(sIdle);
AP.Master = FALSE;
fnThrottle_Hold(0.05);
nDoneFlag = 1; //Aviso de que não completou com sucesso a lição
bCrashCheck = FALSE;
nDelayTime = 0; // Temo decorrido de cada passo
nDelayTime2 = 0; // Um outro tempo
nSequence = 99; // Ajusta a sequência
bDoBad1 = 0;
bDoBad2 = 0;
bDoBad3 = 0;
bDoBadTotal = 0;
bCheat = FALSE;
cpoffice.NAVGPSSwitch = 0;
for nLooper = 0 to 41 do
    ErrorFlag[nLooper] = 0; // Inicializando array em 0
endfor;
for nLooper = 0 to 4 do
    DoneTest[nLooper] = 0; // Não são necessários tantos testes
// Inicializando array em 0
endfor;
Writeln ("Fim da inicialização");
endfunction;
```

# ////////// O CÓDIGO //////////

Code

***/\* Variáveis comuns de inicialização que devem ser acionadas cada vez que um novo loop no simulador é solicitado, juntamente com várias outras ocorrências. \*/***

***// Vôo encerrado.***

```
if (bFinished) then
return;
endif;
```

```
bWOW = acMe.OnGround;          // aeronave no solo
```

```
if (AP.Master) then           //Piloto Automático desligado.
AP.Master = FALSE;
endif;
```

```
nLocDev = posHere.BearingTo (kRwy35Lat,kRwy35Lon);
nLocDev = nLocDev - posHere.MagneticVariation;
nDistance2Thresh = posHere.DistanceTo (kRwy35Lat,kRwy35Lon);
```

***// Variáveis de tempo***

```
nDeltaTime          = fltThisFlight.ElapsedTime - nElapsedTime;
nElapsedTime        = fltThisFlight.ElapsedTime;           // Tempo decorrido de simulação
nTimer              = nTimer + nDeltaTime;
nToleranceTimer     = nToleranceTimer + nDeltaTime;
```

***// Os Objetos ENGINE devem ser iniciados antes da função fnSetEngines ser chamada.***

```
for nLooper = 0 to (nEngineCount-1) do
    ecEngines[nLooper] = cpOffice.EngineController(nLooper); // Objeto controlador dos motores
    eMotors[nLooper] = acMe.Engine(nLooper);
endfor;
```

***// Esses rádios devem ser iniciados antes de serem usados***

```
VHFNAV1              = acMe.VORRadio(1);
VHFNAV2              = acMe.VORRadio(2);
ADF1                 = acMe.ADFRadio(1);
Comm1                = acMe.COMRadio(1);
vorSelect             = acMe.VORRadio(nVORSelect);
nLocalizer            = vorSelect.CourseDeviation;
nGlideSlope           = vorSelect.GSDeviation;
nDMEDistance          = vorSelect.DMEDistance;
```

***// Monitoramento de teclas de interação com o usuário***

```
if (bUserKey1) then Writeln ("User Key 1");endif;
if (bUserKey2) then Writeln ("User Key 2");endif;
```

***// Função para manusear a aceleração, caso necessário***

```
fnThrottle_Manage;
```

***//Mais variáveis para uso futuro***

```
nAltAGL              = acMe.AbsoluteAGLAltitude;
nAltitude             = acMe.PressureAltitude;
nIASNow               = acMe.IndicatedAirspeed;
nVSNOW               = acMe.VerticalSpeed;
nPitch                = acMe.Pitch;
nRoll                 = acMe.Bank;
posHere               = acMe.Position;
nYaw                  = (acMe.Heading - posHere.MagneticVariation);
if (nYaw < 0) then nYaw = nYaw + 360;
endif;
```

***// Cálculo de taxas e razões de movimentos.***

```
nTurnRate            = Diff360 (nYawn1,nYaw)/nDeltaTime;
nYawn1                = nYaw;
```

***// Decrement quiet timers if appropriate***

```
if (nQuietTime > 0) then nQuietTime = nQuietTime - nDeltaTime; endif;
if (nQuietTime2 > 0) then nQuietTime2 = nQuietTime2 - nDeltaTime; endif;
```

***// Enquanto estivermos aqui, cheque para ter certeza de que estamos dentro das tolerâncias***

```
fnToleranceStart;
if (bToleranceCheck) then
    fnCheckTolerances;
endif;
```

***//Colisão***

```
if (bWOW and bCrashCheck) and (not bRunwayCheck) then
    nState = CaseCrashed;
    writeln("***** CRASHED! *****");
endif;
```



# ////////// ÁRVORE SWITCH / CASE //////////

## switch (nState)

```

case CaseInit:                                     // Início dos casos. Retarda a simulação em 10 segundos
Writeln ("CaseInit");
  if (nElapsedTime > 10) then
    nState = CaseDoOnce; // Após estabilizada a simulação, em 10 segundos, mudar para CaseDoOnce
  endif;
endcase;
case CaseDoOnce:
  Writeln ("CaseDoOnce");
  nState = CaseTalk1; // Mudar nState para CaseTalk1
endcase;
case CaseWait:
  Writeln ("CaseWait");
  // não faz nada, somente espera...
endcase;
case CaseWaitForUserKey1:
  Writeln ("CaseWaitForUserKey1");
  // não faz nada, somente espera o usuário acionar ctrl-[
endcase;
case CaseWaitForUserKey2:
  Writeln ("CaseWaitForUserKey2");
  // não faz nada, somente espera o usuário acionar ctrl-]
endcase;
case CaseMovinOn:
  Writeln("CaseMovinOn");
  if (nMoveOnTo == CaseTimeOut) then
    nState = nMoveOnTo;
  else
    fnSQ_ClearAll;
    fnToleranceChecking(FALSE);
    fnSQ_ErrorMsg(swMovinOn,0);
    fnSQ_Play("MovingOn");
    fnSQ_Pause(2);
    fnSQ_Tolerance(4);
    fnSQ_ChangeState(CaseMovinOn2);
    fnSQ_Synchronize;
  endif;
endcase;
case CaseMovinOn2:
  Writeln("CaseMovinOn2");
  fnMsgRestoreNormal;
  fnSQ_ChangeState(nMoveOnTo);
endcase;
case CaseTimeOut:
  Writeln("CaseTimeOut");
  bToleranceCheck = FALSE;
  fnSQ_ClearAll;
  fnSQ_ErrorMsg(swTookTooLong,0);
  fnSQ_Play("EndFlight");
  fnSQ_Pause(10);
  fnSQ_ChangeState(CaseEnd);
  fnSQ_Synchronize;
endcase;
case CaseCrashed:
  writeln("***** CRASHED! *****");
  fnToleranceChecking(OFF);
  bCrashCheck = FALSE;
  fnSQ_ClearAll;
  fnSQ_ErrorMsg(swCrashed,0);
  fnSQ_Pause(3);
  fnSQ_ChangeState(CaseEnd);
  fnSQ_Synchronize;
endcase;
case CaseEnd:
  Writeln ("CaseEnd");
  fnClearMsg;
  fnSQ_ClearALL;
  bFinished = TRUE;
  fltThisFlight.FlightAnalysis;
  fltThisFlight.End(nDoneFlag);
endcase;

```

**case CaseTalk1:****// Após os 10 segundos de retardo inicial, a aventura pula para cá.**

```

Writeln ("CaseTalk1"); // escreve "CaseTalk1" na tela de impressão
AP.Heading = 191; // ajusta o rumo do piloto automático em 191
fnResetToleranceFlags; // aciona a função fnResetToleranceFlags e retorna
nAltitudeTarget = 2000; // define altitude desejada de 2000 pés
nAltitudeToleranceMax = 100; // define variação máx de altitude 100 pés
nAltitudeToleranceMin = -100; // define variação min de altitude 100 pés
nHeadingTarget = 190; // rumo alvo
nHeadingToleranceMax = 10; // define variação máx de rumo em 10 graus
nHeadingToleranceMin = -10; // define variação min de rumo 10 graus
nSpeedTarget = 100; // velocidade alvo 100 Kts
nSpeedToleranceMax = 10; // define variação máx de velocidade 10 Kts
nSpeedToleranceMin = -10; // define variação min de velocidade 10 Kts
nPitchTarget = 0; // pitch alvo 0
nPitchToleranceMax = 10; // tolerância máxima de pitch 10°
nPitchToleranceMin = -5; // tolerância mínima de pitch -5°
nThrottleTarget = 1; // aceleração alvo 1 (100%)
nThrottleToleranceMax = 0.1; // tolerância máxima para aceleração 0.1
nThrottleToleranceMin = -0.1; // tolerância mínima para aceleração -0.1
nRPMTarget = 2400; // RPM alvo 2400 rpm
nRPMTargetMax = 150; // tolerância máxima RPM -150
nRPMTargetMin = -150; // tolerância mínima RPM -150
// o restante segue o raciocínio até agora....
nRollTarget = 0; // Rolagem
nRollToleranceMax = 15;
nRollToleranceMin = -10;
nLocalizerTolerance = 0.5; // Localizador
nGlideSlopeTolerance = 0.4; // GlideSlope
nToleranceTimeWarn = 20;
nToleranceTimeErr = 20;
nVORSelect = 1;
bTextWarnings = TRUE;
bAudioWarnings = FALSE;
bToleranceTimeCheck = FALSE;
nMoveOnTo = CaseTimeOut; // se o tempo esgotar , encerre a lição
fnToleranceChecking(TRUE); // aciona essa função já em TRUE
fnSQ_NormalMsg(s1,0); // exibe a mensagem S1 (em constantes)
fnSQ_Play("PPCR-1"); //You're ready for the next big step
fnSQ_Play("PPCR-2"); //Complete all the maneuvers satisfactorily
fnSQ_Play("PPCR-3"); //
fnSQ_Play("PPCR-4"); //I want you to take off and climb to
fnSQ_Play("PPCR-5"); //Complete all the maneuvers satisfactorily
fnSQ_Play("PPCR-6"); //You'll enter the pattern on the downwind
fnSQ_Play("PPCR-7"); //I won't do any flying for you
fnSQ_Play("PPCR-8"); //I'll prompt you just once on each maneuver
fnSQ_Play("PPCR-9"); //On the other hand, if you don't get it
fnSQ_Play("PPCR-10"); //Ready? Let's go!
fnSQ_ChangeState(CaseTakeoff); // passa o caso agora para CaseTakeoff
fnSQ_Synchronize;
endcase;

```

Esses valores de tolerância não são necessários em suas aventuras ou lições. Você só vai precisar de tolerâncias e limites caso esteja planejando uma aventura como esse voo, a fim de avaliar a capacidade de se manter dentro de limites. Nessa checagem de voo, caso saia dos limites, a aventura se encerra.

PPCR-1 e os outros são arquivos de som que estão localizados na pasta [raiz]/sound/lessons, já com caminho definido na constante strWavBase.

**case CaseTakeoff:****//Preparando para decolar**

```

Writeln ("CaseTakeoff");
bReminder1 = TRUE;
bDoOnce = TRUE;
fnThrottle_Manual;
// Define as coordenadas barométricas da pista de decolagem
nPoint[0,LAT] = 47.49979;
nPoint[0,LON] = -122.7554;
nPoint[1,LAT] = 47.5;
nPoint[1,LON] = -122.7559;
nPoint[2,LAT] = 47.48573;
nPoint[2,LON] = -122.7695;
nPoint[3,LAT] = 47.48552;
nPoint[3,LON] = -122.7690;
fnSQ_NormalMsg(s11,0); // Exibe a mensagem S11
fnSQ_Play("PPCR-11"); //Takeoff and climb to 2000 feet
fnSQ_ToleranceFlag(TF_HEADING,10); // Avisos de tolerância...
fnSQ_ToleranceFlag(TF_THROTTLE,10);
fnSQ_ToleranceFlag(TF_ROLL,10);
fnSQ_ToleranceFlag(TF_PARKINGBRAKE,10);
fnSQ_ChangeState(CaseTakeoff_1); // Muda estado, caso o anterior tenha sido...
fnSQ_Synchronize; // ... executado sem erros.
endcase;

```

**case CaseTakeOff\_1: //decolagem: aguardando o piloto acelerar**

```

Writeln ("CaseTakeOff_1");
// Caso a aceleração dos motores seja iniciada, passamos para o próximo caso.
if (ecEngines[0].Throttle > (nThrottleTarget + nThrottleToleranceMin)) then
    nState = CaseTakeOff_2;
endif;
endcase;

```

**case CaseTakeOff\_2: //decolagem: avançando na pista e elevando o bico da aeronave**

```

Writeln ("CaseTakeOff_2");
// Caso a aeronave não esteja no solo e nos limites da pista, acionarão as tolerâncias de pista
if not bWow then fnTolerance(TF_RUNWAY,OFF,0);
endif;
// Caso velocidade em relação ao solo > 1 Kts, avalie...
if (nIASNow > 1) and (bReminder1) then
    bReminder1      = FALSE;
    bRunwayCheck    = TRUE;
    bCrashCheck     = TRUE;
    bWithinTolerance = FALSE;
    nTimer = 0;
endif;

// Caso velocidade em relação ao solo > 65 Kts e pitch não ajustado,...
if (nIASNow > 65) and (not bPitchCheck) then
    nPitchTarget    = 10;
    bPitchCheck     = TRUE;
    bWithinTolerance = FALSE;
endif;

// Caso dentro das tolerâncias e o pitch checado e aprovado, checando...
if (bWithinTolerance) and (bPitchCheck) then //...rumo, aceleração rolagem e pitch.
    bRunwayCheck = FALSE;
    nTimer = 0;
    nManTime = 0;
    nState = CaseClimb1;
endif;
endcase;

```

A intenção de se fazer as aventuras em casos é basicamente em avaliar a cada período de tempo ou de etapas se a aventura está dentro de determinadas tolerâncias. Nessa checagem de voo, a intenção é avaliar sua capacidade em se manter dentro das tolerâncias determinadas nas constantes do script.

**case CaseClimb1: //Suba para 2000'**

```

Writeln ("CaseClimb1");
//desative a checagem do pitch para a aeronave poder começar a subir sem ser interrompida
if (bPitchCheck) then if (nAltitude >= ((nAltitudeTarget + nAltitudeToleranceMin)-200)) then
    bPitchCheck = FALSE;
endif; endif;
if ((nAltitude >= (nAltitudeTarget - 25)) or (nTimer > 180)) and (not bAltitudeCheck) then
    bAltitudeCheck = TRUE;
    bPitchCheck    = FALSE; // Should be off already, but just to be safe.
    bThrottleCheck = FALSE;
    bWithinTolerance = FALSE;
    fnSQ_NormalMsg(s11a,0);
    fnTolerance(TF_SPEED,ON,60);
    nTimer = 0;
endif;
if (not bSpeedCheck) then if (fnAtSpeed) then fnTolerance(TF_SPEED,ON,0); endif; endif;
if (bWithinTolerance) and (bSpeedCheck) and (bAltitudeCheck) then nManTime = nManTime + 1;
endif;
if (bWithinTolerance) and (nManTime >= 5) then
    nState = CaseClimb1_1;
endif;
endcase;

```

O objetivo nessa caso é atingir 2000` e vamos avaliar isso.  
 Poção 1: atingir altitude mínima numa velocidade normal.  
 Opção 2: Está muito lento, com pitch em 10° e atinge 2000` antes de 3 minutos (180 segundos).

**case CaseClimb1\_1:**

```

Writeln ("CaseClimb1_1");
strAString = "";
if (nRollMax <> 0) then concat(strAString,sBankSteep); endif;
if (nRollMin <> 0) then concat(strAString,sBankShallow); endif;
if (nAltitudeMax <> 0) then concat(strAString,sAltHigh); endif;
if (nAltitudeMin <> 0) then concat(strAString,sAltLow); endif;
if (nSpeedMax <> 0) then concat(strAString,sIASHigh); endif;
if (nSpeedMin <> 0) then concat(strAString,sIASLow); endif;
if (nHeadingMax <> 0) then concat(strAString,sHeadingSoon); endif;
if (nHeadingMin <> 0) then concat(strAString,sHeadingLate); endif;
if (nPitchMax <> 0) then concat(strAString,sPitchHigh); endif;
if (nPitchMin <> 0) then concat(strAString,sPitchLow); endif;
if (bWarnRunway) then concat(strAString,sOffRunway); endif;
if (strAString == "") then
    fnSQ_NormalMsg(sGoodJob,0);
    fnSQ_Play("PPCRGOODJOB");
    fnSQ_Pause(2);
    fnSQ_ChangeState(CaseTurn1);
    fnSQ_Synchronize;
else
    Writeln (fn1Str1Num("nYaw = ",nYaw));
    fnSQ_ErrorMsg(strAString,0);
    fnSQ_Play("PPCRFAIL");
    fnSQ_Pause(5);
    fnSQ_ChangeState(CaseEnd);
    fnSQ_Synchronize;
endif;
endcase;

```

**case CaseTurn1:****//Curva à esquerda seguindo rumo 090**

```

Writeln ("CaseTurn1");
fnResetToleranceFlags;
AP.Heading      = 090;
nHeadingTarget  = 090;
bHeadingCheck   = FALSE;
bSpeedCheck     = TRUE;
nRollTarget     = -20;
bRollCheck      = FALSE;
nManTime        = 0;
bAltitudeCheck  = TRUE;
fnSQ_NormalMsg(s12,0);
fnSQ_Play("PPCR-12");           //Faça curva À ESQUERDA para 090
fnSQ_ToleranceFlag(TF_ROLL,8);
fnSQ_ChangeState(CaseTurn1_1);
fnSQ_Synchronize;
EndCase;           // Caso a curva tenha sido feita adequadamente, segue-se o próximo caso...

```

**case CaseTurn1\_1:****//Curva normal à esquerda**

```

Writeln ("CaseTurn1_1");
if (nYaw <= (nHeadingTarget + 20)) and (bRollCheck) and (not bHeadingCheck) then
    bRollCheck= FALSE;
endif;
if (nYaw <= (nHeadingTarget + nHeadingToleranceMax)) and (not bHeadingCheck) then
    bHeadingCheck= TRUE;
    nTimer = 0;
endif;
if (bHeadingCheck) and (not bRollCheck) and (nTimer > 5) then
    nRollTarget      = 0;
    bRollCheck       = TRUE;
    bWithinTolerance = FALSE;
    nTimer           = 0;
endif;
if (bWithinTolerance) and (bHeadingCheck) and (bRollCheck) then nManTime = nManTime + 1; endif;
if (bWithinTolerance) and (nManTime >= 5) then
    nState = CaseTurn1_2;
endif;
endcase;

```

**case Caseturn1\_2:****//Avalia o trabalho do piloto na curva**

```

Writeln ("CaseTurn1_2");
strAString = "";
if (nRollMax <> 0) then concat(strAString,sBankSteep); endif;           // rolagem máxima
if (nRollMin <> 0) then concat(strAString,sBankShallow); endif;         // rolagem mínima
if (nAltitudeMax <> 0) then concat(strAString,sAltHigh); endif;          // altitude máxima
if (nAltitudeMin <> 0) then concat(strAString,sAltLow); endif;           // altitude mínima
if (nSpeedMax <> 0) then concat(strAString,sIASHigh); endif;             // velocidade máxima
if (nSpeedMin <> 0) then concat(strAString,sIASLow); endif;              // velocidade mínima
if (nHeadingMax <> 0) then concat(strAString,sHeadingSoon); endif;       // curso máximo
if (nHeadingMin <> 0) then concat(strAString,sHeadingLate); endif;       // curso mínimo

if (strAString == "") then           // Caso tudo tenha transcorrido de acordo...
    fnSQ_NormalMsg(sGoodJob,0);      // mensagem de OK
    fnSQ_Play("PPCRGOODJOB");        // som de OK
    fnSQ_Pause(2);
    fnSQ_ChangeState(CaseTurn2);     // passamos para caseturn2
    fnSQ_Synchronize;
else
    // caso contrário...
    Writeln (fn1Str1Num("nYaw = ",nYaw));
    fnSQ_ErrorMsg(strAString,0);      // mensagem de erro
    fnSQ_Play("PPCRFAIL");            // som de erro
    fnSQ_Pause(5);
    fnSQ_ChangeState(CaseEnd);        // passamos para caso de encerramento (CaseEnd)
    fnSQ_Synchronize;
endif;
endcase;

```



**case CaseTurn2:****//Curva 360 graus — iniciando**

```

Writeln ("CaseTurn2");
fnResetToleranceFlags;
bSpeedCheck          = TRUE;
bAltitudeCheck        = TRUE;
bHeadingCheck         = FALSE;
bRollCheck            = FALSE;
nRollTarget           = -45;
nManTime              = 0;
nTimer               = 0;
bPassGo              = FALSE;
bDoOnce              = TRUE;
fnSQ_NormalMsg(s13a,0);
fnSQ_Play("PPCR-13");           //Hora de dar um volta completa
fnSQ_Pause(3);
fnSQ_Play("PPCR-14");           //Faça uma curva de 360 graus para esquerda
fnSQ_ToleranceFlag(TF_ROLL,10);
fnSQ_ChangeState(CaseTurn2_1);
fnSQ_Synchronize;
endcase;

```

**case CaseTurn2\_1:****//Curva 360 graus — continuando na curva**

```

Writeln ("CaseTurn2_1");           // Tenha certeza de fazer toda a curva
if (nYaw > (nHeadingTarget + nHeadingToleranceMax)) and (not bPassGo) then bPassGo = TRUE;
endif;
if (Diff(nYaw,(kHdg1 + 30 + nTurnRate + 5)) < 10) and (bPassGo) then
// 30 graus + razão da curva + 5 graus para morosidade dos sistemas.
    bRollCheck= FALSE;
    fnSQ_NormalMsg(s13b,0);
    else
    strAString = "Target = ";
    concat (strAString,(kHdg1 + 30 + nTurnRate + 5));
    concat (strAString,", current = ");
    Writeln (fn1Str1Num(strAString,nYaw));
endif;

if (bPassGo) and (nYaw < (nHeadingTarget + nHeadingToleranceMax)) and (not bHeadingCheck) then
// Estamos dentro das tolerâncias — inicie checagem do rumo
    bHeadingCheck          = TRUE;
    bWithinTolerance        = FALSE;
    nTimer = 0;
endif;

if (bHeadingCheck) and (not bRollCheck) and (nTimer > 5) then
    nRollTarget            = 0;
    bRollCheck             = TRUE;
    bWithinTolerance        = FALSE;
    nTimer                 = 0;
endif;

if (bWithinTolerance) and (bHeadingCheck) and (bRollCheck) then nManTime = nManTime + 1; endif;
if (bWithinTolerance) and (nManTime >= 5) then
    nState = CaseTurn2_3;
endif;
endcase;

```

**case CaseTurn2\_3:****//Avaliando**

```

Writeln ("CaseTurn2_3");
strAString = "";
if (nRollMax <> 0) then concat(strAString,sBankSteep); endif;
if (nRollMin <> 0) then concat(strAString,sBankShallow); endif;
if (nAltitudeMax <> 0) then concat(strAString,sAltHigh); endif;
if (nAltitudeMin <> 0) then concat(strAString,sAltLow); endif;
if (nSpeedMax <> 0) then concat(strAString,sIASHigh); endif;
if (nSpeedMin <> 0) then concat(strAString,sIASLow); endif;
if (nHeadingMin <> 0) then concat(strAString,sHeadingLate); endif;
if (nHeadingMax <> 0) then concat(strAString,sHeadingSoon); endif;
if (strAString == "") then
    fnSQ_NormalMsg(sGoodJob,0);
    fnSQ_Play("PPCRGOODJOB");
    fnSQ_Pause(2);
    fnSQ_ChangeState(CaseTurn3);
    fnSQ_Synchronize;
else
    Writeln (fn1Str1Num("nYaw = ",nYaw));
    fnSQ_ErrorMsg(strAString,0);
    fnSQ_Play("PPCRFAIL");
    fnSQ_Pause(5);
    fnSQ_ChangeState(CaseEnd);
    fnSQ_Synchronize;
endif;
endcase;

```

**case CaseTurn3: //Seguindo na direção de Tacoma Narrows**

```

Writeln ("CaseTurn3");
fnResetToleranceFlags;
bSpeedCheck          = TRUE;
bAltitudeCheck        = TRUE;
nManTime              = 0;
nTimer               = 0;
nRollTarget           = 20;
AP.Heading            = 110;
nHeadingTarget        = 110;
fnSQ_NormalMsg(s15a,0);
fnSQ_Play("PPCR-15a"); // Allright, nice job so far, I want you to turn to 120
fnSQ_ToleranceFlag(TF_HEADING,10);
fnSQ_NormalMsg(s15b,0);
fnSQ_ChangeState(CaseCruise1);
fnSQ_Synchronize;
endcase;

```

**case CaseCruise1: // zzzz zzzz zzzz**

```

Writeln("CaseCruise1");
strAString = fn1Str1Num("Bearing = ",nLocDev);
if (nDistance2Thresh < 8) then
    fnSQ_Play("PPCR-16"); // I want you to set up for entering the pattern
    bAltitudeCheck= FALSE;
    fnSQ_NormalMsg(s20,0);
    fnSQ_Play("PPCR-20"); // Descend to 1300 feet
    fnSQ_ChangeState(CaseTurn4);
    fnSQ_Synchronize;
endif;
endcase;

```

**case CaseTurn4: //Virar para direção do aeroporto**

```

Writeln ("CaseTurn4");
strAString = fn1Str1Num("Bearing = ",nLocDev);
if (nDistance2Thresh < 4) then
    //KTIW
    nPoint[1,LAT]          = 47.27474;
    nPoint[1,LON]          = -122.5772;
    nPoint[2,LAT]          = 47.27470;
    nPoint[2,LON]          = -122.5766;
    nPoint[3,LAT]          = 47.26111;
    nPoint[3,LON]          = -122.5790;
    nPoint[0,LAT]          = 47.26115;
    nPoint[0,LON]          = -122.5796;
    AP.Heading             = 170;
    nHeadingTarget         = 170;
    bHeadingCheck          = FALSE;
    bRollCheck             = FALSE;
    bSpeedCheck            = FALSE;
    bWithinTolerance       = FALSE;
    fnSQ_NormalMsg(s19,0);
    fnSQ_Play("PPCR-19"); // Turn RIGHT to heading 180
    fnSQ_ChangeState(CaseLanding);
    fnSQ_Pause(2);
    fnSQ_Pause(2);
    fnSQ_Pause(2);
    fnSQ_Pause(2);
    fnSQ_Pause(2);
    fnSQ_NormalMsg(sLand,0);
    fnSQ_ResetTimer;
    fnSQ_Synchronize;
endif;
endcase;

```

**case CaseLanding:****//Pousando**

```

Writeln ("CaseLanding");
if not bRunwayCheck then fnTolerance(TF_RUNWAY,ON,0); endif;
if (nTimer > 480) then
    //O tempo está se alongando. Pare agora.
    strAString = "";
    concat(strAString,sTookTooLong);
    concat(strAString,sFailed);
    fnSQ_Play("PPCRFAIL");
    fnSQ_Pause(5);
    fnSQ_ChangeState(CaseEnd);
    fnSQ_Synchronize;
endif;
if (bWOW) then
    bRunwayCheck                = TRUE;
    bDoOnce                     = TRUE;
    bLandedOffRunway            = TRUE;
    bTaxiedOffRunway            = FALSE;
    bStoppedOffRunway           = TRUE;
    nTouchDownHeading           = nYaw;
    nState                      = Caselanding_1;
endif;
endcase;

```

**case CaseLanding\_1:****//Tenha certeza de que eles estejam na pista**

```

writeln("CaseLanding_1");
if (bWOW) and (bDoOnce) then
    bLandedOffRunway = bWarnRunway;
    fnSQ_ToleranceFlag(TF_SPEED,15);
    nTimer           = 0;
    nSpeedTarget      = 0;
    nSpeedToleranceMax = 5;
    nSpeedToleranceMin = 0;
    bDoOnce           = FALSE;
endif;
if not bTaxiedOffRunway then bTaxiedOffRunway = bWarnRunway; endif;
if (IsStopped)then
    bStoppedOffRunway = bWarnRunway;
    fnSQ_ClearAll;
    fnSQ_ChangeState(CaseLanding_2);
    fnSQ_Synchronize;
endif;
endcase;

```

**case CaseLanding\_2:****//Avaliando o Pouso**

```

writeln("CaseLanding_2");
if (bLandedOffRunway or bTaxiedOffRunway or bStoppedOffRunway) then
    fnSQ_NormalMsg(sFailed,0);
    fnSQ_Play("PPCRFAIL");
    fnSQ_Pause(5);
    fnSQ_ChangeState(CasePostFlight);
    fnSQ_Synchronize;
else
    nDoneFlag = 3; //envie aviso indicando sucesso e emissão do certificado
    fnSQ_NormalMsg(sCongrats,0);
    if abs(nTouchDownHeading - 350) < 20 then
        fnSQ_Play("PPCRSUCCESS");
        fnSQ_Pause(5);
        fnSQ_ChangeState(CasePostFlight);
        fnSQ_Synchronize;
    else
        fnSQ_Play("PPCRDING1");
        fnSQ_Pause(5);
        fnSQ_ChangeState(CasePostFlight);
        fnSQ_Synchronize;
    endif;
endif;
endcase;

```

**case CasePostFlight:****//FIM DO VÔO**

```

writeln("CasePostFlight");
fnClearMsg;
AP.Master = FALSE;
fnThrottle_Manual;
fnSQ_ChangeState(CaseEnd);
fnSQ_Synchronize;

```

endcase;

endswitch;

fnMsgDisplay;

**endmodule.**



E chegamos ao final de nosso voo. A bibliografia básica desse guia é o SDK para FS 2004, que pode ser encontrado no site da Microsoft. Você pode encontrar uma tradução completa do guia — exatamente no molde do original, no site da AEROVIRTUAL ([http://www.aerovirtual.org/arquivos/docs/ABLGuide\\_BR\\_2004.zip](http://www.aerovirtual.org/arquivos/docs/ABLGuide_BR_2004.zip)). Espero que o objetivo de dar uma instrução básica e inicial de como criar as aventuras tenha sido atingido.

Juntamente com esse guia, você irá encontrar três arquivos de scripts:

1. modelo01.abl: o exemplo 01 desse guia, funcionando.
2. modelo02.abl: o segundo modelo, também funcionando (Private PilotCheckride comentado) e
3. matriz.abl: modelo básico que pode ser usado para você criar suas aventuras ou lições, usando os modelos padrões do MSFS da Microsoft. Já contém espaço para constantes, variáveis e têm também a maioria das funções utilizáveis nas lições e aventuras do FS.

Finalmente, agradeço a Felipe Paraizo de Lima da Aerovirtual pelo seu sempre presente apoio e pelo sempre dedicado e muito competente trabalho dele e de toda a equipe da Aerovirtual; apoio sem o qual com certeza esse guia não teria saído da idéia. Bons voos a todos e que tenhamos em breve muitas aventuras em território nacional. Valeu e até a próxima.

Ivan Sinigaglia Nunes Pereira

MaxFlight@ivan.med.br





## Referências:

### **Bibliografia:**

1. Microsoft SDK 2004 (Microsoft)
2. Guias de programação em VB (Internet)

### **Programas:**

1. EditPad Lit: (em português): SuperDownloads ([www.superdownloads.com.br](http://www.superdownloads.com.br))  
<http://superdownloads.ubbi.com.br/download/redir.html?ID=14716&Ico=exe&URL=http|download.jpgsoft.com/editpad/br/SetupEditPadLiteBR.exe.html>
2. Adobe Audition: Adobe  
<http://www.brasil.adobe.com/products/audition/main.html>

### **Aventuras:**

1. Perfect Flights  
[www.fs2000.org](http://www.fs2000.org)
2. FSAdventures  
[www.fsadventures.net](http://www.fsadventures.net)

*Caso tenha baixado apenas esse arquivo texto, sem os modelos, poderá baixá-los em <http://www.ivan.med.br/Modelos.zip>*