



Carlos H. P. Rodrigues (Caique)
www.turbocode.com.br



Definindo o termo “journaling”



- É um tipo de log
- Descreve toda alteração DML (insert/update/delete) ocorrida no banco
- É baseado em transações





- “Log” : Armazenar o conteúdo anterior e o atual de determinadas tabelas e campos possibilitando auditoria.
- “Journaling” : Descrever todas as alterações ocorridas em uma transação.





- Log

- Auditoria de alterações em tabelas
- Apenas tabelas pré-determinadas
- Apenas campos pré-determinados
- Não importa em que transação ocorreu
- Conteúdo anterior e atual dos campos afetados
- Necessário : Usuário / data / hora

- Journaling

- Histórico transacional do banco
- Todas as tabelas do banco
- Todos os campos afetados na transação
- É exccencial transação e ordem das alterações
- Apenas o novo conteúdo dos campos afetados.
- Opcional : Usuário / data / hora





- Backup “a quente” (cada transação esta sendo gravada)
- Restaurar todas as operações DML efetuadas até determinada transação
- Avaliar o custo de uma transação
- Replicação entre bancos





- Cada tabela possui uma ou mais triggers em função das operações a serem monitoradas (insert, update, delete)
- Cada trigger possui um conjunto de “If” verificando se cada campo monitorado teve seu conteúdo alterado.
- As informações são distribuídas em 2 ou mais tabelas, dependendo a forma de implementação do log.



Exemplo de tabelas de Log



```
/* Log de Tabelas */
```

```
CREATE TABLE LOG$TABLES (  
    ID NUMERIC(18,0) NOT NULL PRIMARY KEY,  
    TABLE_NAME VARCHAR(67) CHARACTER SET UNICODE_FSS NOT NULL,  
    OPERATION VARCHAR(1) NOT NULL,  
    DATE_TIME TIMESTAMP NOT NULL,  
    USER_NAME VARCHAR(67) NOT NULL) ;
```

```
/* Log de Chaves */
```

```
CREATE TABLE LOG$KEYS (  
    TABLE_ID NUMERIC(18,0) NOT NULL,  
    KEY_FIELD VARCHAR(67) CHARACTER SET UNICODE_FSS NOT NULL,  
    KEY_VALUE VARCHAR(255) CHARACTER SET UNICODE_FSS) ;
```

```
/* Log de Campos */
```

```
CREATE TABLE LOG$FIELDS (  
    TABLE_ID NUMERIC(18,0) NOT NULL,  
    FIELD_NAME VARCHAR(67) CHARACTER SET UNICODE_FSS NOT NULL,  
    OLD_VALUE VARCHAR(255) CHARACTER SET UNICODE_FSS,  
    NEW_VALUE VARCHAR(255) CHARACTER SET UNICODE_FSS) ;
```

```
/* Log de Campos Blob */
```

```
CREATE TABLE LOG$BLOB_FIELDS (  
    TABLE_ID NUMERIC(18,0) NOT NULL,  
    FIELD_NAME VARCHAR(67) CHARACTER SET UNICODE_FSS NOT NULL,  
    OLD_CHAR_VALUE VARCHAR(10000) CHARACTER SET UNICODE_FSS,  
    NEW_CHAR_VALUE VARCHAR(10000) CHARACTER SET UNICODE_FSS,  
    OLD_BLOB_VALUE BLOB,  
    NEW_BLOB_VALUE BLOB) ;
```



Exemplo de Trigger de Log



```
CREATE TRIGGER LOG$CAD_ENTIDADES_AU FOR CAD_ENTIDADES
ACTIVE AFTER UPDATE POSITION 32767 as
declare variable TABLE_ID integer;
begin
    TABLE_ID = gen_id(LOG$TABLE_ID,1);

    insert into LOG$TABLES (ID, TABLE_NAME, OPERATION, DATE_TIME, USER_NAME)
        values (:TABLE_ID, 'CAD_ENTIDADES', 'U', 'NOW', user);

    insert into LOG$KEYS (TABLE_ID, KEY_FIELD, KEY_VALUE)
        values (:TABLE_ID, 'KCAD_ENTIDADE', old.KCAD_ENTIDADE);

    if (    (old.KCAD_ENTIDADE is null      and new.KCAD_ENTIDADE is not null)
        or (old.KCAD_ENTIDADE is not null and new.KCAD_ENTIDADE is null      )
        or (new.KCAD_ENTIDADE is not null and old.KCAD_ENTIDADE is not null
            and new.KCAD_ENTIDADE <> old.KCAD_ENTIDADE)) then
        insert into LOG$FIELDS (TABLE_ID, FIELD_NAME, OLD_VALUE, NEW_VALUE)
            values (:TABLE_ID, 'KCAD_ENTIDADE', old.KCAD_ENTIDADE, new.KCAD_ENTIDADE);

    if (    (old.CODIGO is null      and new.CODIGO is not null)
        or (old.CODIGO is not null and new.CODIGO is null      )
        or (new.CODIGO is not null and old.CODIGO is not null
            and new.CODIGO <> old.CODIGO)) then
        insert into LOG$FIELDS (TABLE_ID, FIELD_NAME, OLD_VALUE, NEW_VALUE)
            values (:TABLE_ID, 'CODIGO', old.CODIGO, new.CODIGO);

    if (    (old.NOME is null      and new.NOME is not null)
        or (old.NOME is not null and new.NOME is null      )
        or (new.NOME is not null and old.NOME is not null
            and new.NOME <> old.NOME)) then
        insert into LOG$FIELDS (TABLE_ID, FIELD_NAME, OLD_VALUE, NEW_VALUE)
            values (:TABLE_ID, 'NOME', old.NOME, new.NOME);

end;
```





- Criar um campo para transação

```
CREATE TABLE LOG$TABLES (  
    ID NUMERIC(18,0) NOT NULL PRIMARY KEY,  
    TID NUMERIC(18,0) NOT NULL, -- campo para armazenar a transação  
    TABLE_NAME VARCHAR(67) CHARACTER SET UNICODE_FSS NOT NULL,  
    OPERATION VARCHAR(1) NOT NULL,  
    DATE_TIME TIMESTAMP NOT NULL,  
    USER_NAME VARCHAR(67) NOT NULL) ;
```

- Alterar trigger para gravação da transação usando a variável de contexto **CURRENT_TRANSACTION**

```
insert into LOG$TABLES  
(ID, TID, TABLE_NAME, OPERATION, DATE_TIME, USER_NAME)  
values  
(:TID, CURRENT_TRANSACTION, 'CAD_ENTIDADES', 'U', 'NOW', USER);
```



Problemas da abordagem : Journaling = Log



- Crescimento rápido do banco
 - Cada tabela afetada na transação gera um registro
 - Cada campo de cada tabela gera um registro
- Performance
 - Recuperar uma transação pode envolver a consulta de centenas ou milhares de registros
- Exportação
 - Complexidade de descrever uma transação em arquivo externo ao banco mantendo sua coerência
 - “Anexar” campos BLOB à transação
- Importação
 - Complexidade na geração de DML e manuseio de BLOB





- Segurança
 - gravar as transações externas ao banco principal
- Performance
 - uma transação deve ser obtida com uma consulta simples (evitar joins), preferencialmente a um único registro.
- Exportação
 - um único arquivo deve descrever toda a transação
 - Campos BLOB anexo ao arquivo da transação
- Importação
 - Facilidade na geração de DML e manuseio de BLOB





- Padrão de mercado
 - recomendado pelo W3C
- Criado em formato “texto”
- Projetado para armazenar e transportar dados
- É uma linguagem de marcação
 - conjunto de códigos (marcações) aplicados a um texto ou a dados, com a finalidade de adicionar informações específicas sobre este, tornando-se auto descritivo.
- Flexível, não possui marcações pré-definidas
 - Marcações criadas conforme a necessidade da descrição dos dados (crie suas próprias marcações).





- Dados devem ser declarado entre marcações
- Marcações são chamadas de tag (etiqueta)
- Uma tag, é um nome que descreve o dado que contém e deve ser declarada entre os sinais de “menor que” e “maior que”:

<nome_da_tag>

- Toda tag deve possuir uma tag de finalização, representada por uma barra antes de seu nome :

</nome_da_tag>





- Uma Tag vazia (que não possua um dado associado) pode ser representada por uma barra antes do sinal de “maior que”, dispensando o uso da tag de finalização :

<nome_da_tag/>

- A tag é “case sensitive” (diferencia maiúsculas /minúsculas)
- Não deve possuir espaços em seu nome





- Uma tag pode possuir um ou mais atributos, que descrevem propriedades específicas da mesma ou dos dados a que se referem.
- Atributos são representados por um par nome/valor, declarados após o nome da tag, o valor deve sempre estar entre aspas :

<nome_da_tag a1="v1" a2="v2" ... aN="vN">





- Dados não podem conter caracteres especiais
- São considerados caracteres especiais “<” e “&”
- Caracteres especiais devem ser convertidos para uma notação específica
- “>” não é considerado caractere especial, mas a boa prática sugere converte-lo para notação específica
- Por se tratar de um arquivo texto, não são permitidos caracteres de controle da tabela ASCII (códigos inferiores a 32)





Existem 5 notações pré-definidas para caracteres especiais :

Caractere	Descrição	Notação
<	menor que	<
>	maior que	>
&	'e comercial'	&
'	apóstrofo	'
"	aspas	&aquote;





- `<?xml version="1.0"?>`
 - Identifica um documento texto no formato XML, deve ser a primeira tag do documento.
 - Pode possuir outros atributos como como “encoding”, indicando a página de código utilizada nos dados
- `<!-- comentário -->`
 - Indica um comentário, apenas descritivo, não representando dados





- `<![CDATA[conteúdo]]>`
 - Indica que os dados de uma Tag não devem ser analisados pelo parser XML.
 - Permite o uso de caracteres “<” e “&” sem codificação especial.
 - Não são permitidos caracteres de controle da tabela ASCII (códigos inferiores a 32)





- Declaração do formato XML
 - `<?xml version="1.0"?>`
- Elemento Raiz – Node/Document Root
 - Tag que conterá todos os demais elementos descritos no documento
- Elementos filhos – Child Nodes
 - Tags dentro do nó principal (Node/Document Root) ou de outros elementos filhos, propiciando um documento estruturado em “árvore”



Exemplo de XML



```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

```
<mensagem>
```

```
  <para>Participantes</para>
```

```
  <de>FDD8</de>
```

```
  <assunto>Ajudem o FB crescer</assunto>
```

```
  <texto>Ao final do evento, serão aceitas  
doações monetárias espontâneas de qualquer  
importância, as quais serão repassadas à  
Firebird® Foundation.
```

```
  </texto>
```

```
</mensagem>
```



Estrutura XML para o journaling



- Document Root
<transaction Id="9" start="data/hora" end="data/hora" username="nome">
- Lista de tabelas
<tables count="9">
- Descrição de uma tabela
<table Name="tabela" sequence="9" updatekind="I/U/D" timestamp="data/hora">
- Lista de campos chave
<PrimaryKey count="9">
- Campo da chave
<pkfield Name="campo">
- Lista de campos
<fields count="9">
- Descrição do campo
<field Name="campo" IsNull="T">
- Lista de campos do tipo blob
<blobfields count="9">
- Descrição de um campo da tabela
<blobfield Name="campo" IsNull="T">





- O formato XML não aceita caracteres especiais da tabela ASCII (códigos abaixo de 32)
- Campos BLOB podem conter caracteres com códigos abaixo de 32

Workarround : Codificar campos BLOB, mesmo que binários em uma representação textual aceita pelo XML.





- **Base64** é um método para codificação de dados para transferência na Internet (*codificação MIME para transferência de conteúdo*) .
- É utilizado frequentemente para transmitir dados binários por meios de transmissão que lidam apenas com texto, como por exemplo para enviar arquivos anexos por email.
- Padrão definido pela RFC989 (obsoleta)

Fonte:<http://pt.wikipedia.org/wiki/Base64>



A mágica por traz da codificação Base64



- A codificação Base64 possui uma tabela pré definida de 64 caracteres ASCII :
A-Z, a-z, 0-9, +, /
- Cada 3 octetos (1 octeto = 8 bits) de informação (24 bits), são convertidos em pacotes de 6 bits, gerando o máximo de 64 valores binários diferentes.
- Cada valor binário é representado por um caractere da tabela Base64.
- Cada 3 octetos codificados geram 4 caracteres textuais ($24 \text{ bits} / 6 \text{ bits} = 4 \text{ octetos}$)





- Criação de Triggers em todas as tabelas do banco responsável por :
 - Gravar Informações sobre a tabela
 - Gravar a primary key de qualquer registro alterado (insert, update, delete)
 - Gravar o novo valor de cada campo alterado
 - Codificar campos BLOB em Base64
- Procedure para exportação da transação





- Universal Triggers
- Database Triggers
- Context variables
- User / system context variables
- Monitoring tables
- Distinct from
- Global temporary table (GTT)
- Execute statement
- UDF – User defined function





- Permite que uma trigger lide com diversas operações (insert, update, delete)

- Sintaxe :

```
CREATE TRIGGER name FOR table  
[ACTIVE | INACTIVE]  
{BEFORE | AFTER} <multiple_action>  
[POSITION number]  
AS trigger_body
```

```
<multiple_action> ::= <single_action> [or <single_action> [or <single_action>]]  
<single_action> ::= {INSERT | UPDATE | DELETE}
```





- Permite que uma trigger seja disparada em operações do banco

- Sintaxe :

```
{CREATE | RECREATE | CREATE OR ALTER}  
[ACTIVE | INACTIVE]  
ON <event>  
[POSITION number]  
AS trigger_body
```

```
<event> ::= CONNECT  
          | DISCONNECT  
          | TRANSACTION START  
          | TRANSACTION COMMIT  
          | TRANSACTION ROLLBACK
```





- **CURRENT_TRANSACTION**
 - Retorna o identificador da transação atual
 - Tipo : inteiro ; escopo : DSQL, PSQL
- **INSERTING / UPDATING / DELETING**
 - Determina o tipo de operação sendo executada
 - Tipo : lógica ; escopo : PSQL, apenas em triggers.
- **Outras variáveis de contexto**
 - CURRENT_CONNECTION
 - SQLCODE / GDSCODE
 - ROW_COUNT



User / system context variables



- `RDB$SET_CONTEXT(<namespace>, <variable>, <value>)`
 - Permite setar o valor de uma variável em um contexto
 - `RDB$GET_CONTEXT(<namespace>, <variable>)`
 - Permite recuperar o valor de uma variável em um contexto
- `<namespace> ::= 'SYSTEM' || 'USER_SESSION' || 'USER_TRANSACTION'`
- O namespace 'SYSTEM' possui variáveis pré-definidas, read-only.
 - As variáveis do namespace 'USER_SESSION' são visíveis/acessíveis no contexto da sessão (conexão)
 - As variáveis do namespace 'USER_TRANSACTION' são visíveis/acessíveis no contexto da transação
 - Máximo de 1000 variáveis por sessão ou transação





- Permite monitorar a atividade no servidor e do banco de dados conectado
 - MON\$TRANSACTIONS : Monitora as transações iniciadas no banco de dados
- Outras tabelas de monitoramento
 - MON\$DATABASE (o banco conectado)
 - MON\$ATTACHMENTS (conexões)
 - MON\$STATEMENTS (comandos preparados)
 - MON\$CALL_STACK (pilha utilizada pelo PSLQ ativo)
 - MON\$IO_STATS (estatísticas de I/O)
 - MON\$RECORD_STATS (estatísticas a nível de registros)
 - MON\$MEMORY_USAGE (uso de memória – ODS 11.2)
 - MON\$CONTEXT_VARIABLES (variáveis de contexto – ODS 11.2)



Distinct from



- Faz a comparação de 2 valores mas não considera valores nulos como distintos

- Sintaxe :

<valor> IS [NOT] DISTINCT FROM <valor>

```
if ( old.CAMPO is distinct from new.CAMPO ) then
```

Equivale a :

```
if (      ( old.CAMPO is null      and new.CAMPO is not null )
    or ( old.CAMPO is not null and new.CAMPO is null      )
    or ( new.CAMPO is not null and old.CAMPO is not null
        and new.CAMPO <> old.CAMPO ) ) then
```





- Criar tabelas temporárias

Sintaxe :

```
CREATE GLOBAL TEMPORARY TABLE
```

```
( <fields> ...
```

```
) [ON COMMIT <DELETE | PRESERVE> ROWS]
```

- ON COMMIT DELETE ROWS (default)
 - Remove as linhas da tabela ao finalizar a transação
- ON COMMIT PRESERVE ROWS
 - Remove as linhas da tabela ao finalizar a conexão



Execute statement



- Converte uma string em DSQL e executa no banco corrente ou em um banco externo
- Sintaxe :

```
[FOR] EXECUTE STATEMENT <query_text> [( <input_parameters> )]  
[ON EXTERNAL [DATA SOURCE] <connection_string>]  
[WITH AUTONOMOUS | COMMON TRANSACTION]  
[AS USER <user_name>]  
[PASSWORD <password>]  
[ROLE <role_name>]  
[WITH CALLER PRIVILEGES]  
[INTO <variables>]
```

```
<input_parameters> ::=  
    <named_parameter>  
    | <input_parameters>, <named_parameter>  
<named_parameter> ::=  
    <parameter name> := <expression>
```



UDF – User defined function



- São funções que não são internas (building functions) ao Firebird, mas definida em módulos separados.
- Estes módulos são DLL / SO (Windows / Linux)
- O Firebird vem com duas bibliotecas UDF:
 - IB_UDF (coleção de UDF do InterBase)
 - fbudf (nova coleção de UDF do Firebird)
- Novas bibliotecas de UDF podem ser incorporadas ao Firebird
- Existem UDF's free, pagas, e a possibilidade de criar suas proprias UDF.
- UDF's podem ser criadas em qualquer linguagem que gere DLL (ou SO) e respeite a convenção de chamada da linguagem C.



Registrando a UDF



- Uma biblioteca de UDF como o nome diz pode conter uma ou várias funções.
- Além de possuir o módulo DLL / SO é necessário registrar cada uma das UDF em cada banco que for utiliza-la.
- UDF não declaradas não são acessíveis pelo banco, mesmo que existam outros bancos no mesmo servidor que as utilizem.

- Sintaxe :

```
DECLARE EXTERNAL FUNCTION name [datatype | CSTRING (int)
[, datatype | CSTRING (int) ...]]
RETURNS {datatype [BY VALUE] | CSTRING (int)} [FREE_IT]
[RETURNS PARAMETER n]
ENTRY_POINT 'entryname'
MODULE_NAME 'modulename';
```



Escrevendo uma UDF para BLOB



- UDF para tratamento de campos BLOB, difere de outras UDFs, pelo fato dos parametros recebidos se tratarem de ponteiros para uma estrutura de controle e não aos dados em si.
- A UDF não pode abrir ou fechar um BLOB, mas invoca funções de acesso a este atraves do ponteiro da estrutura de controle.
- Estrutura de controle do BLOB :

```
typedef struct blob {  
    void (*blob_get_segment) ();  
    isc_blob_handle blob_handle;  
    long number_segments;  
    long max_seglen;  
    long total_size;  
    void (*blob_put_segment) ();  
} *Blob;
```

Fonte : [%PathFBSever%\ include\ibase.h]



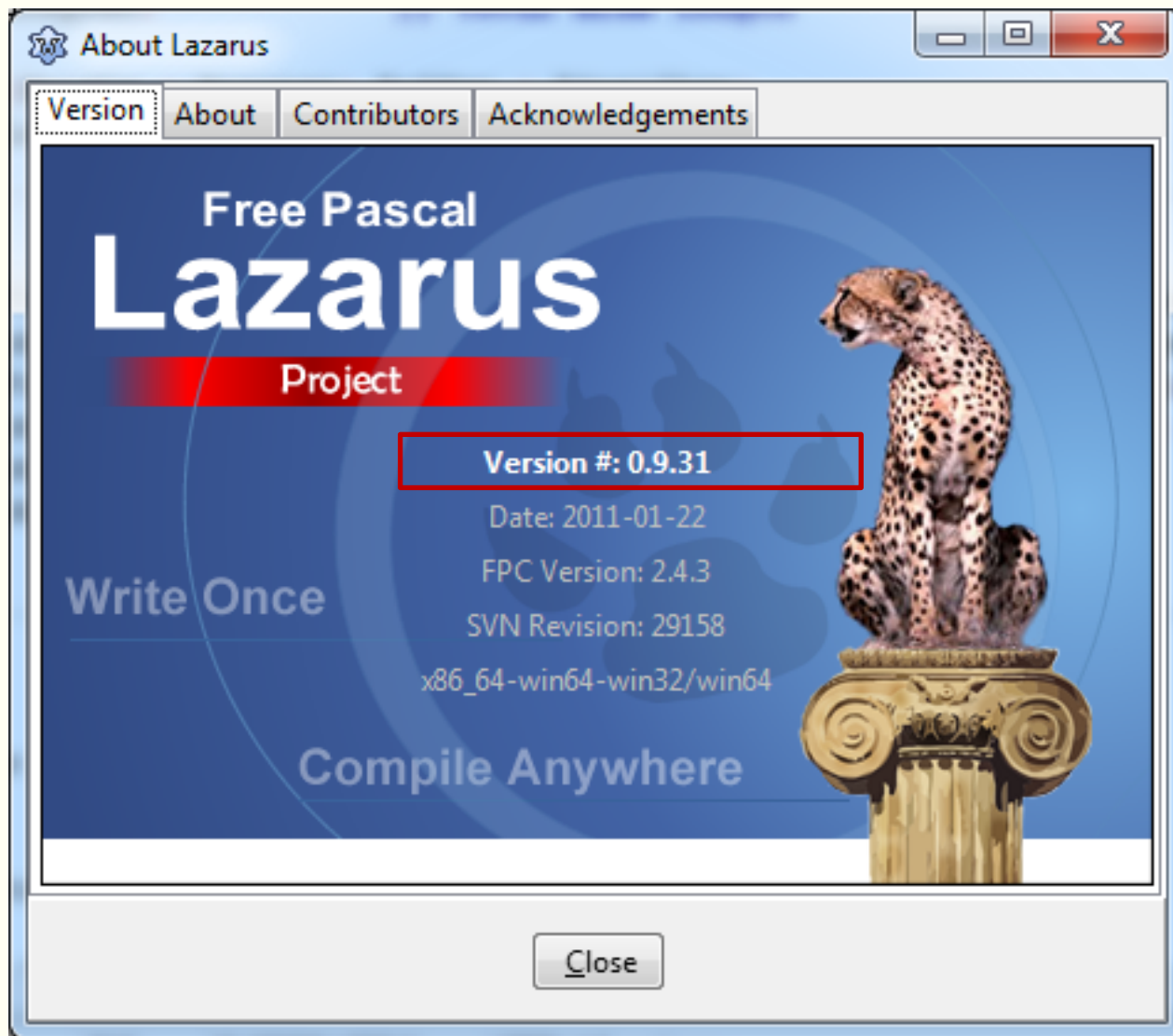
UDF necessárias para o journaling



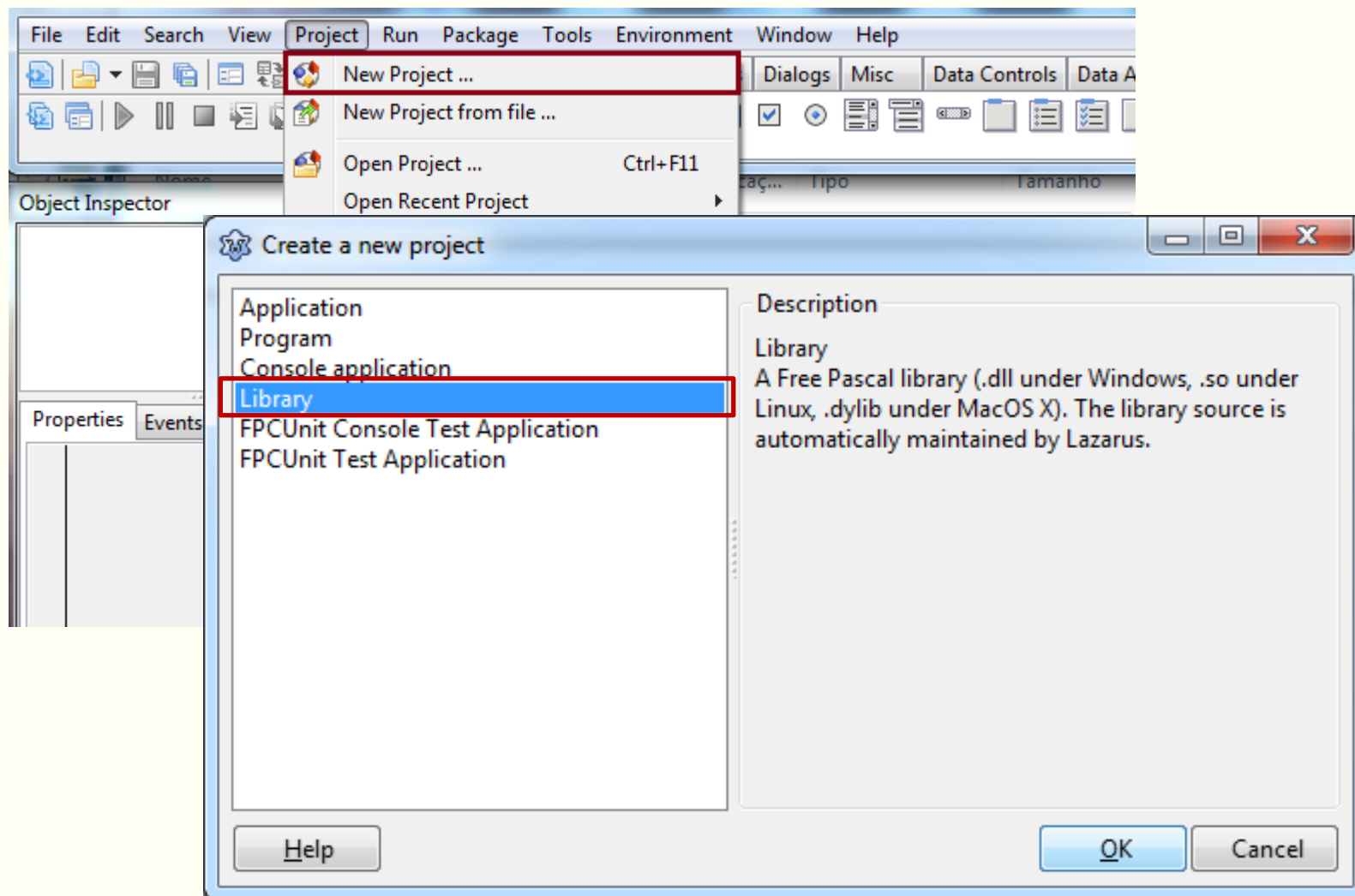
- SaveBlobToFile (AFileName: PAnsiChar; var ABlob: TBLOB)
 - Salva o blob em disco
- AppendBlobToFile (AFileName: PAnsiChar; var ABlob: TBLOB)
 - Acrescenta informações em um arquivo existente
- LoadBlobFromFile (AFileName: PAnsiChar; var ABlob: TBLOB)
 - Le um BLOB do disco
- ImportBlobFromFile(AFileName: PAnsiChar; var ABlob: TBLOB)
 - Le um e paga o BLOB do disco
- EncodeBlobBase64 (var ABlob, ACodedBlob : TBLOB)
 - Codifica o conteúdo de um BLOB em Base64
- DecodeBlobBase64 (var ABlob, ADecodedBlob : TBLOB)
 - Decodifica o Base64 para o conteúdo original



Lazarus – 0.9.31 – dialy build



Criando um novo projeto em Lazarus



Criando um novo projeto em Lazarus



```
Source Editor
*project1.lpr

1  library Project1;
.
.  {$mode objfpc}{$H+}
.
5  uses
.    Classes
.    { you can add units after this };
.
.  {$R *.res}
10
.  begin
.    end.
13
```

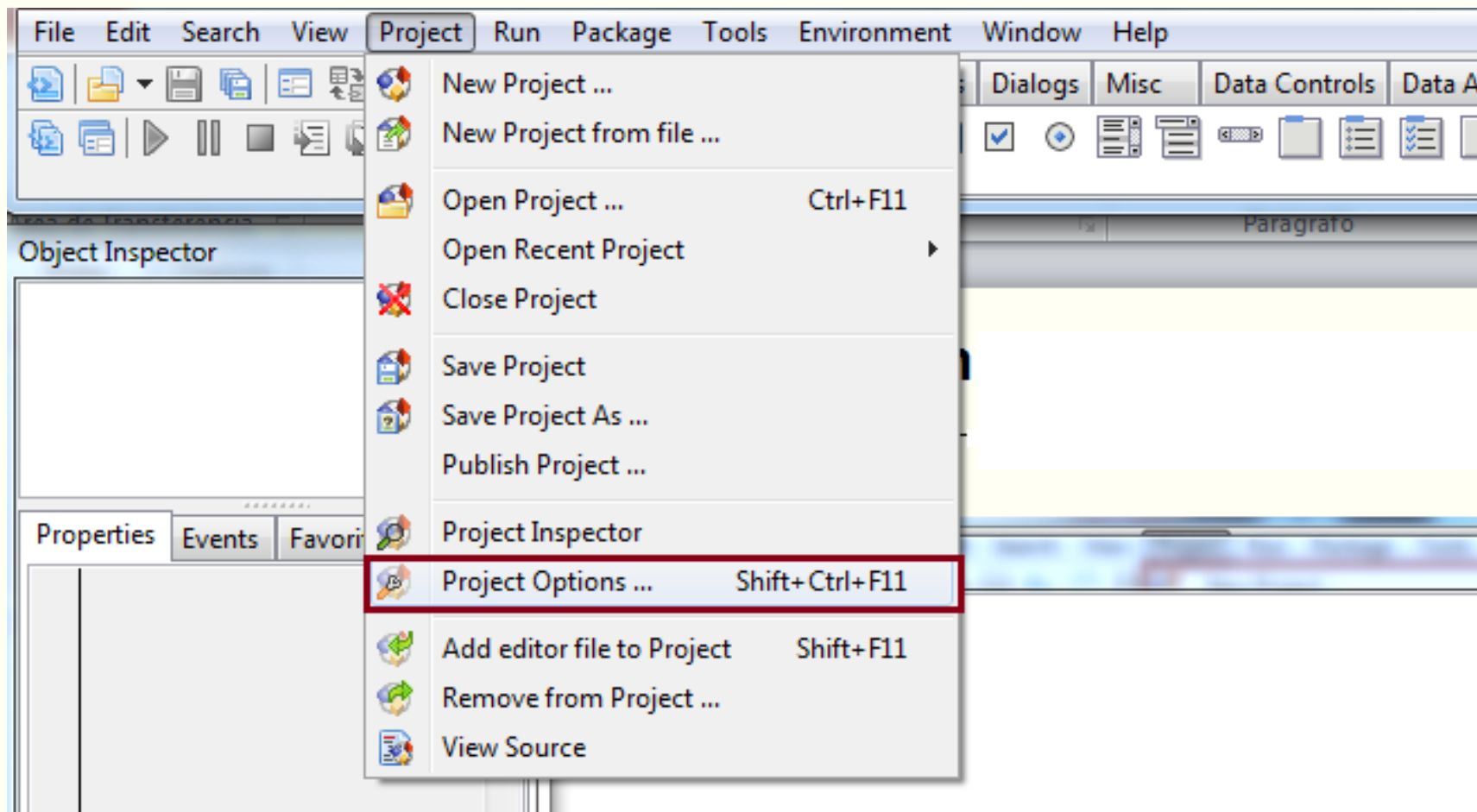
Salvar o Projeto com um nome apropriado

Remover as diretivas padrão

Remover a diretiva de recursos



Configurando opções do projeto em Lazarus



Configurando opções do projeto em Lazarus



Options for Project: sguid

Project Options

- Application
- Forms
- FPDoc Editor
- Session
- Version Info**
- i18n
- Miscellaneous

Compiler Options

- Paths
- Build modes
- Parsing
- Code generation
- Linking
- Verbosity
- Messages
- Other
- Build macros
- Inherited
- Compilation

☒ Include Version Info in executable

Version numbering

Major version: 1 Minor version: 64 Revision: 0 Build: 0

☐ Automatically increase build number

Language options

Language selection: U.S. English Character set: Multilingual

Other info

Key	Value
ProductVersion	
CompanyName	
FileDescription	
FileVersion	1.64.0.0
InternalName	
OriginalFilename	

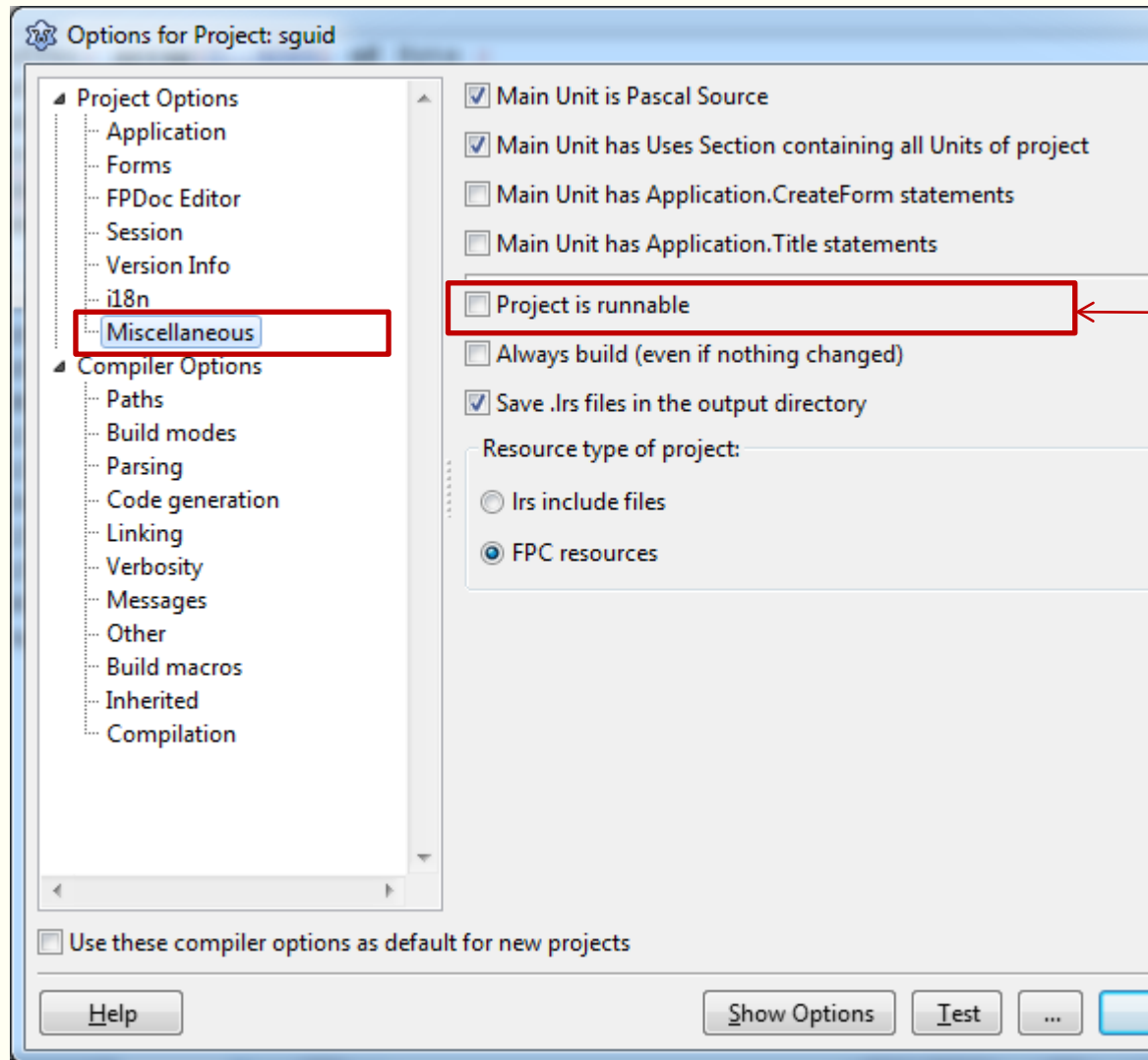
☐ Use these compiler options as default for new projects

Help Show Options Test ... OK

Opcional : Configurar opções de versão



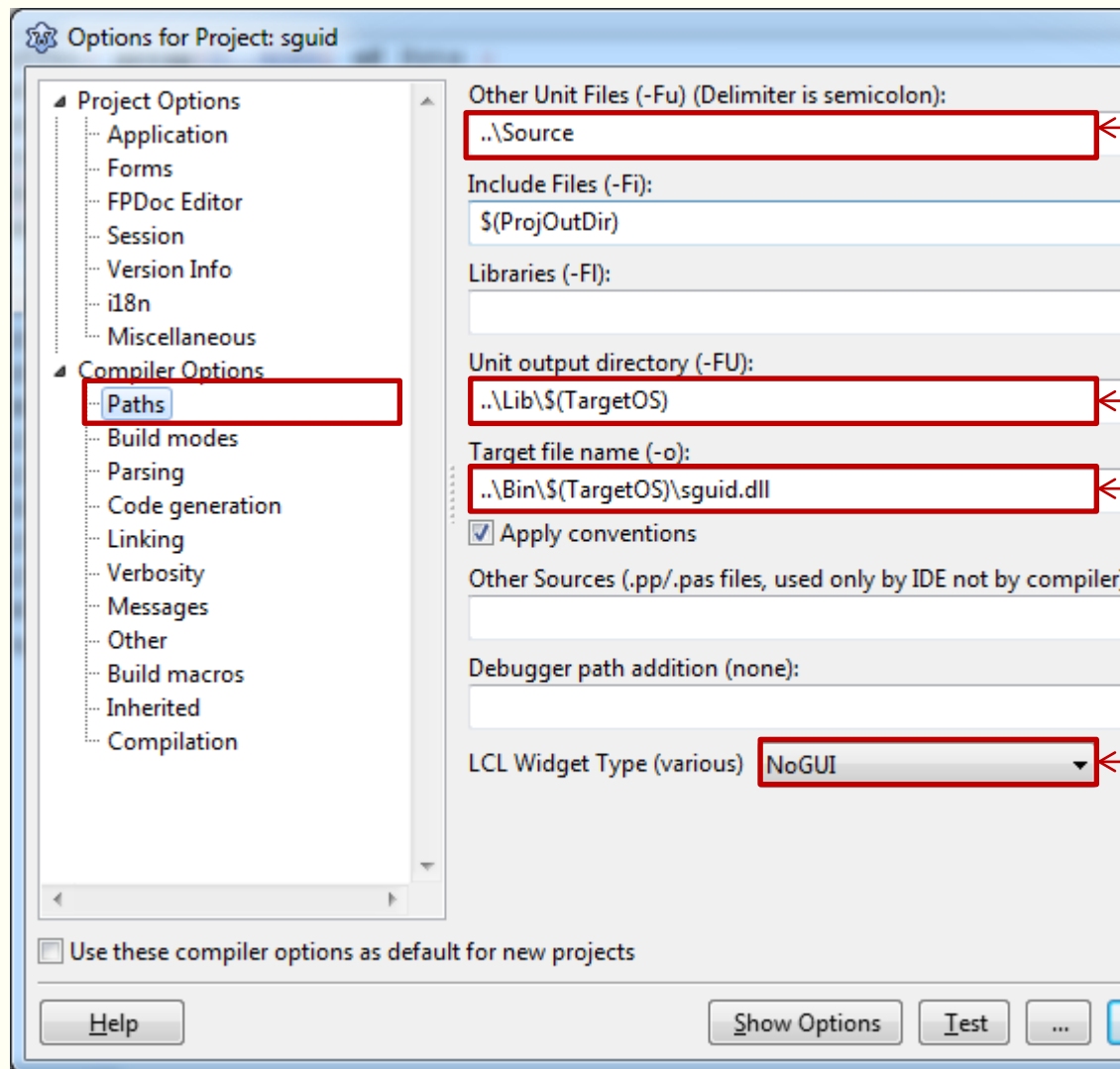
Configurando opções do projeto em Lazarus



Desmarcar a opção
Project runnable –
informa a IDE que este
projeto não é executavel



Configurando opções do projeto em Lazarus



Onde pesquisar as units do projeto

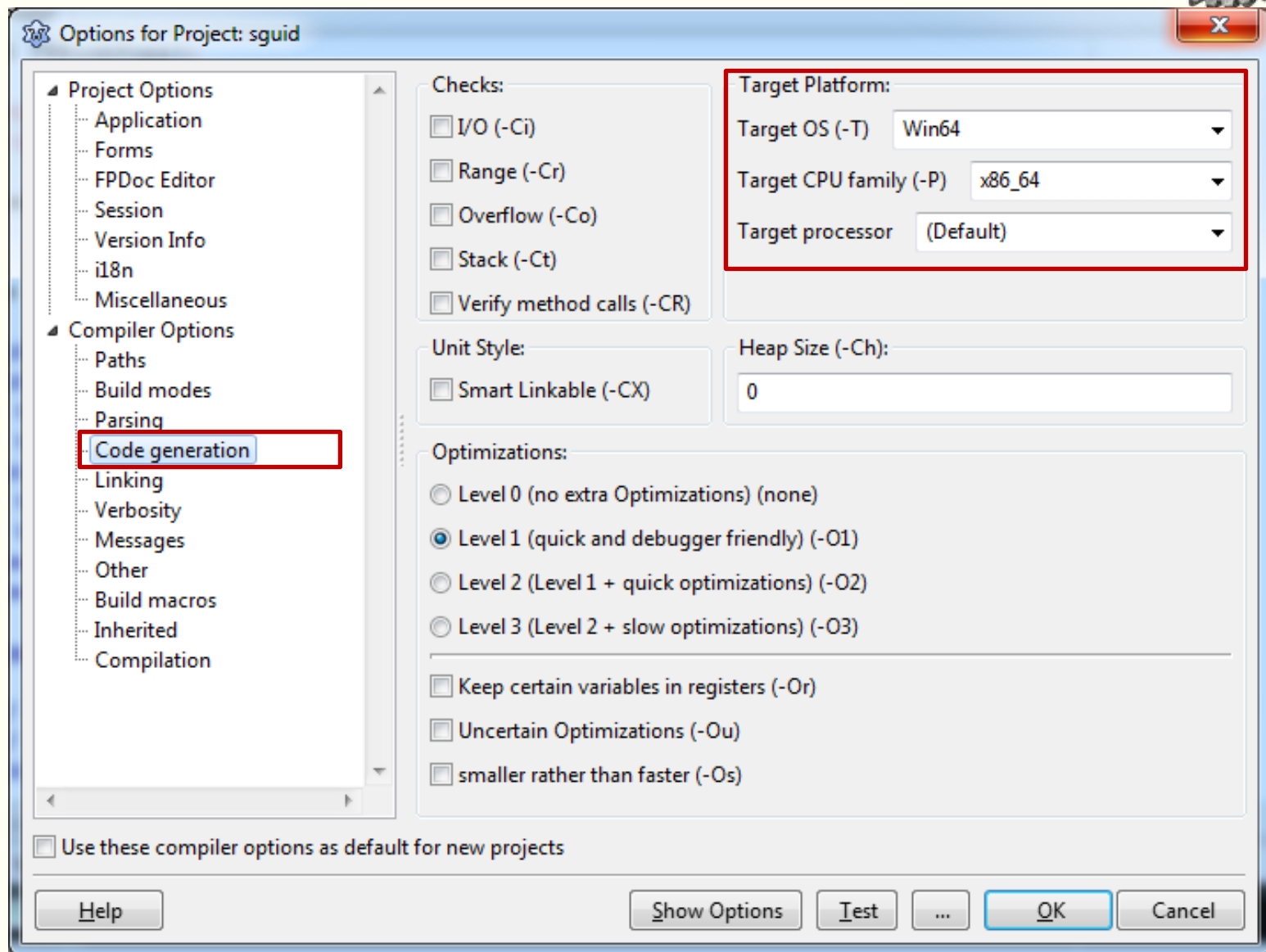
Onde gravar as units compiladas do projeto

Onde gravar o projeto linkado (binário) – especificar a extensão

Indicar que o projeto não possui uma interface gráfica



Gerando a DLL para 64 bits



Exportando as funções UDF



```
sguid
1  library sguid;
.
.
.  uses
.    FBGuidSuppl in '..\Source\FBGuidSuppl.pas',
5    FBBlobSuppl in '..\Source\FBBlobSuppl.pas',
.    GuidSuppl in '..\Source\GuidSuppl.pas' ;
.
.
.  exports
.    FBGuidSuppl.GUIDSTR,
10   FBGuidSuppl.GUIDFMT,
11   FBGuidSuppl.GUID36,
.    FBGuidSuppl.GUID32,
.    FBGuidSuppl.GUID22,
.    FBGuidSuppl.GUIDSTRTO36,
15   FBGuidSuppl.GUIDSTRTO32,
.    FBGuidSuppl.GUIDSTRTO22,
.    FBBlobSuppl.SaveBlobToFile,
.    FBBlobSuppl.AppendBlobToFile,
.    FBBlobSuppl.LoadBlobFromFile,
20   FBBlobSuppl.ImportBlobFromFile,
.    FBBlobSuppl.EncodeBlobBase64,
.    FBBlobSuppl.DecodeBlobBase64 ;
.
.  begin
25  end.
```

Units utilizadas no projeto

Declaração das funções que serão acessíveis externamente

Funções para tratamento de campos BLOB – atentar ao fato do nome das funções exportadas serem “case-sensitive”



Procedure LOGIN



```
CREATE OR ALTER PROCEDURE SYS$LOGIN (  
    PUSERNAME type of SYS$USERLOGIN,  
    PLOCAL_MAC type of SYS$MAC ADDR,  
    PLOCAL_IP type of SYS$IPV6,  
    PSITE_IP type of SYS$IPV6)  
as  
BEGIN  
    RDB$SET CONTEXT ('USER_SESSION', 'SYS$USER_NAME', pUSERNAME ) ;  
    RDB$SET CONTEXT ('USER_SESSION', 'SYS$LOCAL_MAC', pLOCAL_MAC ) ;  
    RDB$SET CONTEXT ('USER_SESSION', 'SYS$LOCAL_IP', pLOCAL_IP ) ;  
    RDB$SET CONTEXT ('USER_SESSION', 'SYS$SITE_IP', pSITE_IP ) ;  
END
```





```
CREATE GLOBAL TEMPORARY TABLE JNL$GTTRANSACTION (  
    JNL$TRANSACTION SEQUENCE    INTEGER NOT NULL,  
    JNL$TRANSACTION                INTEGER NOT NULL,  
    JNL$START                      TIMESTAMP NOT NULL,  
    JNL$TABLES                     INTEGER DEFAULT 0 NOT NULL  
) ON COMMIT DELETE ROWS;
```



Procedure para grava/Obtém dados da Transação



```
CREATE GENERATOR JNL$TRASACTION SEQUENCE ;

CREATE OR ALTER PROCEDURE JNL$GET TRANSACTION SEQUENCE
returns (
    JNL$TRASACTION SEQUENCE integer)
as
declare variable START_TRANSACTION timestamp;
BEGIN

    SELECT MON$TIMESTAMP FROM MON$TRANSACTIONS
    WHERE MON$TRANSACTION_ID = CURRENT_TRANSACTION
    INTO :START_TRANSACTION ;

    SELECT JNL$TRASACTION SEQUENCE FROM JNL$GITTRANSACTION
    WHERE JNL$TRANSACTION = CURRENT_TRANSACTION
        AND JNL$START = :START_TRANSACTION
    INTO :JNL$TRASACTION_SEQUENCE ;
```



Procedure para grava/Obtém dados da Transação (continuação)



```
IF ( JNL$TRASACTION SEQUENCE is NULL ) THEN
    INSERT INTO JNL$GTTRANSACTION
        ( JNL$TRASACTION SEQUENCE,
          JNL$TRANSACTION,
          JNL$START,
          JNL$TABLES
        )
    VALUES
        (
          NEXT VALUE FOR JNL$TRASACTION SEQUENCE,
          CURRENT_TRANSACTION,
          :START_TRANSACTION,
          1
        )
    RETURNING JNL$TRASACTION SEQUENCE
    INTO :JNL$TRASACTION_SEQUENCE ;
ELSE
    UPDATE JNL$GTTRANSACTION
    SET JNL$TABLES = JNL$TABLES + 1
    WHERE JNL$TRASACTION SEQUENCE = :JNL$TRASACTION_SEQUENCE ;

    SUSPEND ;

END^
```



DB-Trigger : Responsável pela exportação



```
CREATE OR ALTER TRIGGER JNL$$EXPORT
ACTIVE ON TRANSACTION COMMIT POSITION 1
as
DECLARE JNL$TRASACTION SEQUENCE INTEGER ;
DECLARE JNL$TRANSACTION INTEGER ;
DECLARE JNL$START          TIMESTAMP ;
DECLARE JNL$TABLES          INTEGER ;
DECLARE xml BLOB SUB_TYPE 1 ;
BEGIN

    SELECT JNL$TRASACTION SEQUENCE, JNL$TRANSACTION, JNL$START, JNL$TABLES
    FROM JNL$GTTRANSACTION
    INTO JNL$TRASACTION SEQUENCE, JNL$TRANSACTION, JNL$START, JNL$TABLES ;

    if ( JNL$TABLES > 0 ) then
        EXECUTE STATEMENT ( 'EXECUTE PROCEDURE JNL$ADD ?, ?, ?, ?, ?, ?, ?' )
            ( JNL$TRASACTION SEQUENCE,
              JNL$TRANSACTION,
              JNL$START,
              CAST ( 'NOW' AS TIMESTAMP ),
              RDB$GET CONTEXT ( 'USER_SESSION', 'SYS$USER_NAME' ),
              JNL$TABLES,
              RDB$GET CONTEXT ( 'SYSTEM', 'DB_NAME' )
            )
        ON EXTERNAL DATA SOURCE RDB$GET CONTEXT ( 'SYSTEM', 'DB_NAME' ) || '.JNL'
        WITH COMMON TRANSACTION ;
    END
```



UDF : Declarando funções de BLOB



```
DECLARE EXTERNAL FUNCTION APPENDBLOBTOFILE
    CSTRING(1024) ,
    BLOB,
    INTEGER
RETURNS PARAMETER 3
ENTRY_POINT 'AppendBlobToFile' MODULE_NAME 'sguid';

DECLARE EXTERNAL FUNCTION ENCODEBLOBBASE64
    BLOB,
    BLOB
RETURNS PARAMETER 2
ENTRY_POINT 'EncodeBlobBase64' MODULE_NAME 'sguid';
```





```
CREATE DATABASE
```

```
'127.0.0.1/30250:PATH\BANCOJNL.JNL'
```

```
USER 'SYSDBA' PASSWORD 'masterkey'
```

```
PAGE_SIZE 4096
```

```
DEFAULT CHARACTER SET ISO8859_1;
```



Banco para Journaling : Declarando UDF s



```
DECLARE EXTERNAL FUNCTION IMPORTBLOBFROMFILE  
  CSTRING(1024) ,  
  BLOB  
  RETURNS PARAMETER 2  
  ENTRY_POINT 'ImportBlobFromFile' MODULE_NAME 'sguid';
```

```
DECLARE EXTERNAL FUNCTION LOADBLOBFROMFILE  
  CSTRING(1024) ,  
  BLOB  
  RETURNS PARAMETER 2  
  ENTRY_POINT 'LoadBlobFromFile' MODULE_NAME 'sguid';
```

```
DECLARE EXTERNAL FUNCTION SAVEBLOBTOFILE  
  CSTRING(1024) ,  
  BLOB ,  
  INTEGER  
  RETURNS PARAMETER 3  
  ENTRY_POINT 'SaveBlobToFile' MODULE_NAME 'sguid';
```





```
CREATE TABLE JNL$TRANSACTIONS (  
    JNL$TRASACTION_SEQUENCE    INTEGER,  
    JNL$TRANSACTION            INTEGER,  
    JNL$START                   TIMESTAMP,  
    JNL$END                     TIMESTAMP,  
    JNL$USERNAME                CHAR(31) CHARACTER SET UNICODE_FSS,  
    JNL$XML                     BLOB SUB_TYPE 1 SEGMENT SIZE 80  
);
```



Banco para Journaling : Procedure para adicionar um Journaling



```
CREATE OR ALTER PROCEDURE JNL$ADD (  
    JNL$TRASACTION_SEQUENCE INTEGER,  
    JNL$TRANSACTION INTEGER,  
    JNL$START TIMESTAMP,  
    JNL$END TIMESTAMP,  
    JNL$USERNAME CHAR(31) CHARACTER SET UNICODE_FSS,  
    JNL$TABLES INTEGER,  
    JNL$SOURCEDB VARCHAR(1024))  
  
AS  
  
DECLARE xml BLOB SUB_TYPE 1 ;  
BEGIN
```



Banco para Journaling : Procedure para adicionar um Journaling (continuação)



```
if ( JNL$TABLES > 0 ) then
  BEGIN
    xml = '<?xml version="1.0" encoding="ISO8859-1"?>'
    || '<transaction id="' || JNL$TRASACTION_SEQUENCE
    || '" start="' || JNL$START
    || '" end="' || CAST( 'NOW' AS TIMESTAMP )
    || '" username="' || COALESCE ( TRIM ( JNL$USERNAME ), '' ) || '">'
    || '<tables count="' || JNL$TABLES || '>'
    || ImportBlobFromFile(
      JNL$SOURCEDB -- path/banco.ext
    || '-' || JNL$TRASACTION_SEQUENCE || '-' -- -Sequencia_
    || JNL$TRANSACTION || '.JNL' -- Transacao.jnl
    )
    || '</tables>'
    || '</transaction>' ;

  INSERT INTO JNL$TRANSACTIONS
    ( JNL$TRASACTION_SEQUENCE, JNL$TRANSACTION,
      JNL$START, JNL$END, JNL$USERNAME, JNL$XML )
  VALUES
    ( :JNL$TRASACTION_SEQUENCE, :JNL$TRANSACTION,
      :JNL$START, :JNL$END, :JNL$USERNAME, :XML ) ;

END
```

END



Esboço da Trigger responsável pela montagem do XML de cada tabela



```
CREATE OR ALTER TRIGGER JNL$TABELA FOR TABELA
ACTIVE AFTER INSERT or UPDATE or DELETE POSITION 32767
AS
    DECLARE JNL$TRASACTION_SEQUENCE    INTEGER ;
    DECLARE nf SMALLINT;
    DECLARE xml BLOB SUB_TYPE 1 ;
    DECLARE fxml BLOB SUB_TYPE 1 ;
    DECLARE VARIABLE START_TRANSACTION TIMESTAMP ;
BEGIN

    EXECUTE PROCEDURE JNL$GET_TRANSACTION_SEQUENCE
    RETURNING_VALUES JNL$TRASACTION_SEQUENCE ;

    xml = '<table Name="TABELA"'
        || ' sequence="" || NEXT VALUE FOR JNL$TABLE_SEQUENCE || "'
        || ' updatekind="'
        ||     CASE
                WHEN ( inserting ) THEN 'I'
                WHEN ( updating ) THEN 'U'
                ELSE 'D'
            END
        || "' -- updatekind"
        || ' timestamp="" || cast ( 'NOW' as TIMESTAMP ) || ">' ;
```



Esboço da Trigger responsável pela montagem do XML de cada tabela



```
-- prepara xml com dados da primary key para update/delete
IF ( not inserting ) THEN
BEGIN
    xml = xml || '<PrimaryKey count="1">' ;

    IF ( OLD.CHAVE is NULL ) THEN
        xml = xml || '<pkfield Name="CHAVE" IsNull="T"/>' ;
    ELSE
        |
        xml = xml || '<pkfield Name="CHAVE">' || OLD.CHAVE || '</pkfield>' ;
    xml = xml || '</PrimaryKey>' ;
END

-- prepara xml dos campos incluídos/alterado, exceto blob
nf      = 0 ;
fxml    = '' ;

IF ( (inserting and NEW.CHAVE IS NOT NULL)
    or (updating and NEW.CHAVE IS DISTINCT FROM OLD.CHAVE) ) THEN
BEGIN
    nf = nf + 1 ;
    IF ( NEW.CHAVE is NULL ) THEN
        fxml = fxml || '<field Name="CHAVE" IsNull="T"/>' ;
    ELSE
        fxml = fxml || '<field Name="CHAVE"><![CDATA[' || NEW.CHAVE || ']]></field>' ;
    END
END
```



Esboço da Trigger responsável pela montagem do XML de cada tabela



```
-- prepara xml dos campos incluídos/alterado, exceto blob
nf      = 0 ;
fxml    = '' ;

IF (    (inserting and NEW.CHAVE IS NOT NULL)
      or (updating and NEW.CHAVE IS DISTINCT FROM OLD.CHAVE) ) THEN
BEGIN
  nf = nf + 1 ;
  IF ( NEW.CHAVE is NULL ) THEN
    fxml = fxml || '<field Name="CHAVE" IsNull="T"/>' ;
  ELSE
    fxml = fxml || '<field Name="CHAVE">' || NEW.CHAVE || ']]&gt;&lt;/field&gt;' ;
END

IF (    (inserting and NEW.CAMPO IS NOT NULL)
      or (updating and NEW.CAMPO IS DISTINCT FROM OLD.CAMPO) ) THEN
BEGIN
  nf = nf + 1 ;
  IF ( NEW.CAMPO is NULL ) THEN
    fxml = fxml || '&lt;field Name="CAMPO" IsNull="T"/&gt;' ;
  ELSE
    fxml = fxml || '&lt;field Name="CAMPO"&gt;<![CDATA[' || NEW.CAMPO || ']]&gt;&lt;/field&gt;' ;
END

IF ( nf &gt; 0 ) THEN
BEGIN
  xml = xml || '&lt;fields count="' || nf || '"&gt;' || fxml || '&lt;/fields&gt;' ;
  fxml = '' ;
END</pre></div><div data-bbox="898 838 996 946" data-label="Image"><img alt="A stylized green bird, likely a toucan, with a Brazilian flag motif on its body."/></div><div data-bbox="38 956 377 989" data-label="Page-Footer"><p>www.FirebirdDevelopersDay.com.br</p></div><div data-bbox="478 954 516 986" data-label="Page-Footer"><p>62</p></div><div data-bbox="557 954 996 989" data-label="Page-Footer"><p>© 2011 – Carlos H.P. Rodrigues ( Caique )</p></div>
```

Esboço da Trigger responsavel pela montagem do XML de cada tabela



```
-- prepara xml dos campos BLOB incluidos/alterado
nf      = 0 ;
fxml    = '' ;

IF (    (inserting and NEW.BLOB IS NOT NULL)
      or (updating and NEW.BLOB IS DISTINCT FROM OLD.BLOB) ) THEN
BEGIN
    nf = nf + 1 ;
    IF ( NEW.BLOB is NULL ) THEN
        fxml = fxml || '<blobfield Name="BLOB" IsNull="T"/>' ;
    ELSE
        fxml = fxml || '<blobfield Name="BLOB">'
            || '<![CDATA[' || ENCODEBLOBBASE64 ( NEW.BLOB ) || ']]>'
            || '</blobfield>' ;
    END
END

IF ( nf > 0 ) THEN
    BEGIN
        xml = xml || '<blobfields count="' || nf || '>' || fxml || '</blobfields>' ;
        fxml = '' ;
    END

-- finaliza geracao do xml e salva em disco
xml = xml || fxml || '</table>' ;

AppendBlobToFile(
    RDB$GET_CONTEXT( 'SYSTEM', 'DB_NAME' ) -- path/banco.ext
    || '-' || JNL$TRANSACTION_SEQUENCE || '-' -- -Sequencia_
    || CURRENT_TRANSACTION || '.JNL',      -- Transacao.jnl
    xml ) ;

END
```

