DFSCRIPTOR

Summary

Author: *Sergey V. Natarov*
Created on: *18/02/2017*
Updated on: *20/02/2017*
Version: *2.0*

Overview

DataFlex Scriptor (aka DataFlex for Applications, DFS) project was started about 10 years ago as a review of the GoldParser engine. Later, GoldParser project was frozen by author and seems like not supported. Due to number of changes in the DataFlex, original GoldParser DLL is not supported anymore and latest one (as a third-party component) does not implement all features and works slightly buggy.

So, DFScriptor was re-wrote from the ground recently using DataFlex 18.x, GoldParser engine was removed and parser was created using pure DataFlex code and shows acceptable level of the performance.

The main purpose of this project is to provide ability of the Custom Applications to have expandable functionality. Possible areas of usage might be - dynamic menus, custom built-in reports, wide range of the application and data management utilities, application debugging and so on.

When you have many remote sites (installations of the product), sometimes it becomes a pain to manage (support) these sites remotely, especially when site is live and you do not have full access to. Periodically, you need to examine remote system, review remote database, provide some service utilities and updates. In all these cases DataFlex Scriptor might be a solution or at least significantly simplify your life.

Well, possibly at some stage it will be included into TheHammer 3 project.

This is open source project.

Contents

Summary 1
Overview 1
Contents 2
What's New (Comparing to v. 1) 4
  General Changes 4
  Language Changes 4
  Samples 5
  Script Samples 5
  Application Samples 5
DFScriptor File Extensions 6
  Script Source 6
  Script Package 6
  Script Executable 6
  Script PRN 6
  Script Tables 6
DataFlex Script Debugger 7
  Script Page 7
  Scopes Page 7
  Variables Page 7
  Tokens Page 8
  Errors Page 8
  Symbols Page 8
  Charsets Page 8
  Debugger Hot Keys 9
Supported Data Types 10
  Base Types 10
  Extended Types 10
Supported Commands 11
  Scopes 11
  Logical Operations 13
  Methods and Attributes 14

Assign Operations....15
Database....16
Input/Ouptut....19
General Purposes....22
Supported Compiler Directives....23
Parser....24
Pre-Compiler....25
Builder....26
File/Scope Name Record....27
Scope Record....27
Scope Tokens Record....27
Variable/Constant List Record....28
Security Options....29
DataFlex Scriptor High Level Methods....30
Defining Scriptor Instance....30
Loading Script....30
Calling Script....31
Class Description....33
cDFScriptor Hierarhy....33
cDFScriptor Properties....33
cDFScriptor Methods....34
Expanding Possibilities....38
Adding New Command....38
Extending Functionality....38
Project File Reference....39
General Feedback....40

What's New (Comparing to v. 1)

General Changes

1. Built using 100% DataFlex (v. 18+).

2. GoldParser DLL was removed entirely and replaced by DataFlex based parser.

3. Brand new DFS Debugger, with CodeMax control.

4. New script options (Pre-compile and Build).

Language Changes

1. DataFlex tags are now supported ("{}").

2. DataFlex native arrays (Alpha version is parser only).

3. Most of commands now support expressions as parameters.

4. Input/Output commands now support channels.

5. Get/Set commands are implemented (additionally to the Send command).

6. Many new data types added (Alpha version is parser only).

7. Script based procedures and functions.

8. Number of compiler directives are implemented (eg. USE, #INCLUDE, #REM).

9. Objects can be instantiated using Object/End_object commands. Please note - nested objects and object augmentation are not supported by design.

10. New database management command are implemented - LOCK and ATTACH. As well as functionality expanded for all commands to support file list (where applicable) and options (line DF_ALL).

11. Additionally to MOVE command new set added - INCREMENT, DECREMENT, ADD and SUBTRACT.

12. Repeat/Until/Loop statements were implemented.

13. INDICATE command support for the number of built-in indicators (FOUND, SEQEOF, SEQEOL, ERR).

14. IF/ELSE commands are improved to support BEGIN/END blocks and constructions like IF/ELSE IF.

15. ERROR command is added (along with #ERROR directive).

16. Commands STRUCTURE_START, STRUCTURE_END and STRUCTURE_ABORT are implemented.

17. Script security options added to prohibit execution of the SAVE, SAVERECORD, DELETE, SET_ATTRIBUTE, STRUCTURE_START, STRUCTURE_END, STRUCTURE_ABORT, BEGIN_TRANSACTION and END_TRANSACTION commands. By default, security options are ON (execution is allowed).

Samples

Number of script samples and examples of usage provided.

Script Samples

1. ScriptSampleAttributes.ds - demonstrates general operations with DataFlex attributes.

2. ScriptSampleDatabase.ds - demonstrates general DataFlex database related operations.

3. ScriptSampleIfs.ds - demonstrates general logical operations.

4. ScriptSampleInputOutput.ds - demonstrates general input/output operations.

5. ScriptSampleScopes.ds - using different scopes in the scripts.

6. ScriptSampleAll.ds - combines all samples listed above into the single script file.

Application Samples

1. DataFlex Scriptor Console (available as a tab page under DFS Debugger).

2. Dynamic application menu item.

3. Simple in-line Reporting.

DFScriptor File Extensions

Script Source

Script source file should have *.DS extension. This is text file with the script source (DataFlex statements/phrases are expected).

Script Package

Script package should have *.DP extension. This is text file with the script source (DataFlex statements/phrases are expected). File to be included into the script source files using USE or #INCLUDE directives.

Script Executable

*.DE extension is allocated to the pre-compile script file. This file includes script tables (list of the variables, sources, included files etc). Pre-compiled script file does not require compilation and may be executed straight away. This is not mandatory file. DataFlex Scriptor may load and execute both - source (DS) and executable (DE) files.

Script PRN

*.PRN file is an intermediate code listing file. It will be created during pre-compilation process and located at the same directory where script is located. File might be useful for the debugging and review purposes.

Script Tables

This type of the file is provided for the debugging purposes in case DFS Debugger is not used. *.TBL file will be created at the same location along with script and provides developer with debugging information - scopes available, information about scopes and tokens, list of variables created, Scriptor symbols used.

ⓘ **Please note:** **.DE, *.PRN and *.TBL files are not required for the deployment. *.PRN and *.TBL should never be deployed. *.DE should be deployed in case you wish to provide better performance or secure your script code from the user.*

DataFlex Script Debugger

DataFlex Script Debugger is an optional component that is provided to assist you in script writing.

Script Page

Provides script source window, pre-compiler output window, script structure, Application objects and Application database lists.

Script source window is a CodeMax source editor with syntax highlighting. The same component is used in TheHammer project and probably DataFlex Studio.

Pre-compiler output window provides basic information on the Script compilation, build and run processes.

Script structure shows current script structure tree - including source files and available scopes. You may drag and drop methods available into the script source editor.

Application objects shows the current tree of the parent application objects. It may be used to simplify the reference of these objects in the script source. Drag and Drop operations are supported along with Copy/Paste routines.

Application database - provides detailed list of the parent application database. Including, data tables, columns and indexes. Drag and Drop along with Copy/Paste operations are supported as well. You may drag different database elements. It will insert appropriate access method for the selected element (eg. to simplify Find or Save operations writing).

Scopes Page

Provides list of the script scopes (Procedures, Functions, Objects) and pre-processed tokens list along with list of available variables. If variable has "IN" symbol, then this variable should be passed as a parameter. Scope code will be executed using list of these tokens.

Variables Page

This page shows all defined script variables for all scopes, including main script. It includes detailed information on variable (or constant) name, type, scope, dimensions, internal image and current value is applicable.

Tokens Page

Tokens page shows complete list of the parsed and not pre-compiled tokens. Actual processing goes through scopes. This list is provided for the references only and includes Token number, Symbol ID, Image, Source file, Line and column where token is found by parser.

Errors Page

Errors page shows script errors if any. It mostly duplicates pre-compile information window (may be removed at the later versions).

Symbols Page

Symbols are the DataFlex Scriptor IDs allocated to the parsed tokens if found or applicable. This page provided for the information on complete list of the symbols available.

Symbols have a few general types:

1. Token - abstract symbol for the appropriate token found (eg. String or Integer constants, End of Line or End of Script symbols, Directive, Scope, Command and so on.

2. Command - represents recognized DataFlex command in the script (for example, MOVE, FIND, INTEGER etc.).

3. Directive - represents recognized DataFlex Compiler Directives (USE, #INCLUDE, #REM, #ERROR etc).

4. Type - recognized type of the variable defined in the script.

5. Operator - represents DataFlex named operators (GE, GT, CONTAINS etc). Unnamed operators (+, -, * etc.) are recognized like Token type "Operator" symbol (DFT_OPERATOR).

6. Keyword - recognized DataFlex command reserved keyword (RECNUM, CHANNEL, FROM, IS, A etc).

7. Function - recognized DataFlex built-in function (SizeOfArray, Pos, Sin, Trim etc.). These function will be used at the later versions of the scriptor.

Charsets Page

Current parser charsets used to recognize tokens.

Debugger Hot Keys

Hot Key	Description
CTRL+N	Clear Debugger and start new script.
CTRL-O	Clear Debugger and open new saved script (*.DS, *.DE, *.DP)
F2	Save current Script.
CTRL+S	Save current script and pre-compile.
CTRL+B	Build the compiled script (produce *.DE).
F5	Run current script.

ⓘ **Please note:** Other hot keys may be assigned (and changed) using DFS Debugger source editor options. However, Options are not implemented so far in the Alpha version and all options are hardcoded (so may be changed in the code only).

Supported Data Types

Base Types

Base data types supported are *Integer*, *Number*, *Real*, *String* and *Date*. Additionally, *Constants* are supported (DEFINE statement) and database *Tables* as constants (OPEN statement).

Base data types are validated well and converted on the fly.

Number of extra data types are based on the base types (Pointer, Handle, Boolean). These type are treated as an Integer type.

Extended Types

Extended data types are: *DateTime*, *TimeSpan*, *Time*, *Char*, *UChar*, *Short*, *UShort*, *UInteger*, *BigInt*, *UBigInt*, *Address*, *RowID* and *Variant*. These types are supported by parser only for the moment. In the later versions I have intention to implement some of them.

ⓘ **Please note:** Native arrays for the listed types are supported at the parser level only. I have intention to implement native arrays in the script. Structs are not supposed to be implemented on other stages.

Supported Commands

Scopes

WHILE/LOOP

Marks a block of code to be executed repeatedly depending on a condition.

```
While {Boolean-expression}
    ... commands ...
Loop
```

FOR/LOOP

Executes a block of code in an iteration.

```
For {counter} From {start-value} To {stop-value}
    ... commands ...
Loop
```

REPEAT/UNTIL

Executes a repeating block of code.

```
Repeat
    ... commands ...
Until {Boolean-expression}
```

BEGIN/END

To define the beginning and end of a block of commands whose execution is to be controlled by other commands.

```
Begin
    ... commands ...
End
```

FUNCTION/FUNCTION_RETURN/END_FUNCTION

To define a function.

```
Function {function-name} [Global|Overloaded|For Class] [{type} {param} … ] ;
    Returns {return-type}
    ... commands ...
    Function_Return {value}
End_Function
```

PROCEDURE/PROCEDURE_RETURN/END_PROCEDURE

To define a procedure.

```
Procedure [Set] {function-name} [Global|Overloaded|For Class] [{type} {param} … ] ;
    [Returns {return-type}]
    ... commands ...
    Procedure_Return {value}
End_Procedure
```

OBJECT/END_OBJECT

To create an object instance of a class. Objects in the DFScriptor may not be nested or augmented. These commands are used for the instantiation of the single object only.

```
Object {object-name} is a {object-class}
End_Object
```

CLASS/END_CLASS

To create a new DataFlex object class. A class defines a DataFlex object's behavior. DFScriptor does not support classes. These statements are for the parser only.

```
Class {sub-class} Is A {super-class}
:
End_Class
```

SCRIPT_RETURN

Similar to function or procedure return command, this extra command is provided for the Script return feature. So script may be called as a function and value can be returned.

Logical Operations

IF/IFNOT/ELSE

Conditionally controls the execution of one or more lines of code.

```
If|IfNot {BooleanExpression} [{Command}|Begin
    {True Statement}
End]
[Else [{Command}|Begin
    {False Statement}
End]
```

Methods and Attributes

SET

To set the value of an object property.

```
Set property [of object] [item_number] To value
```

GET

To get the value of a function or object property.

```
Get {method-name} [of {object-ID}] [{param1 … paramN}] To {receiving-
variable}
```

SEND

To send a message to an object.

```
Send message [of object] [arguments...]
```

GET_ATTRIBUTE

To retrieve a global, driver, database/connection, table, column, or index attribute.

```
Get_Attribute {attribute} [of {tableHandle|DriverID} ;
    [{databaseHandle} |{columnNum} | {indexNum} [{segmentNum}]]] ;
    To {variable}
```

SET_ATTRIBUTE

To set a global, driver, database/connection, table, column, or index attribute.

```
Set_Attribute {attribute} [of {tableHandle|DriverID} ;
    [{databaseHandle} |{columnNum} | {indexNum} [{segmentNum}]]] ;
    To {variable}
```

SYSDATE

To return the current system date, hour, minute, and second.

```
SysDate {dtDate} [{iHour} [{iMinute} [{iSecond}]]]
```

Assign Operations

MOVE/CALC

To assign a value to a variable. CALC command will be replaced with MOVE during calculations.

```
Move {expression} to {variable}
```

ADD

To perform addition.

```
Add {value} to {variable}
```

SUBTRACT

To perform subtraction.

```
subtract value from variable
```

INCREMENT

Increase the value of an integer variable by one.

```
Increment {variable}
```

DECREMENT

To reduce the value of a variable or database field by one.

```
Decrement {variable}
```

INDICATE

Indicators as a type are not supported. This command may change the state of the built-in only indicators (FOUND, SEQEOF, SEQEOL, ERR).

```
Indicate {Built-in Indicator} {True|False|[as {Logical Expression}]}
```

Database

CLOSE

To close a database table.

```
Close {table}[… {table}] | DF_ALL [ DF_TEMPORARY | DF_PERMANENT ]
```

REREAD

To maintain data integrity in multi user systems by updating record buffer data for any changes made by other users since the data was first brought into the buffers, and to prevent further changes by other users until an unlock command has been executed.

```
Reread [{table} … {table}]
```

LOCK

To prevent other users from making changes to a database while one user is saving a record.

```
Lock
```

UNLOCK

In a multi user environment, to end the locked state of all database files imposed by the lock and reread commands.

```
Unlock
```

CLEAR

To erase all data from the record buffers of one or more database tables. The status of the tables is set inactive.

```
Clear {DF_ALL | table} […{table}]
```

SAVE

To save changes made to a record buffer, including data from parent tables, to the database on disk. Please check security option.

```
Save {child-table} […{child-table}]
```

SAVERECORD

To save changes made to a record buffer to the database on disk without updating fields that relate to other tables with data from those tables. Please check security option.

```
SaveRecord table [...table]
```

DELETE

To remove the record currently in the record buffer from each database table named in the command. Please check security option.

```
Delete {table} […{table}]
```

RELATE

To find parent records matching the data in a child table's record buffer.

```
Relate {child-table} […{child-table}]
```

ATTACH

To move the primary key from each parent table into the matching external keys of the named child database tables.

```
Attach {child-table} [… {child-table}]
```

FIND

To retrieve a record from a database table on disk, and place it in the record buffer. Please check security option.

```
Find LT | LE | EQ | GE | GT  {table} By  Recnum | {IndexNumber}
```

BEGIN_TRANSACTION

To explicitly mark the beginning of a database transaction. Please check security option.

```
Begin_Transaction
```

END_TRANSACTION

To explicitly mark the end of a database transaction. Please check security option.

```
End_Transaction
```

STRUCTURE_START

This command begins creation of a new table or modification of an existing one. Please check security option.

```
Structure_Start tableNum [driverName]
```

STRUCTURE_END

Commits a structure operation, performing restructure on the table data if necessary. Please note - callback is not supported. Please check security option.

```
Structure_End tableNum [restructureOptions]
```

STRUCTURE_ABORT

To terminate a table structure operation without completing it. Please check security option.

```
Structure_Abort tableNum
```

Input/Ouptut

SHOW

To display a string of characters to the screen in a special "character-mode" display box that deactivates it when its Close button is pressed.

```
Show value [... value]
```

SHOWLN

To display a line of characters to the screen in a special "character-mode" display box that deactivates when its Close button is pressed.

```
Showln [value... value]
```

WRITE

To Write the values of one or more variables to a sequential file, device or text field.

```
Write [Channel channelNum] value [ ... value]
```

WRITELN

To Write the values of one or more variables to a sequential file, device, or text field as a line of data.

```
WriteLn [Channel ChannelNum] [value ... value]
```

WRITE_HEX

To write binary data as two-byte hexadecimal numbers.

```
Write_hex [Channel channelNum] value [ ... value]
```

READ

To read "words" of data from a sequential file, device, or text field and move them to one or more variables.

```
Read [Channel {channel-num}] {variable} […{variable}]
```

READLN

To read a line of data from a sequential file, device, text field, or image, and move its "words" into one or more variables.

```
ReadLn [Channel {channelNum}] {variable} […{variable}]
```

READ_BLOCK

To read a block of data of a specified size from a sequential file, device, or text field and move it to a variable.

```
Read_Block [Channel {channel-num}] {variable} {length}
```

READ_HEX

To read binary data as two-byte hexadecimal numbers.

```
Read_Hex [Channel {channel-num}] {variable} [{length}]
```

DIRECT_OUTPUT

To write output to a sequential file, a printer, or the Windows clipboard.

```
Direct_Output [Channel {ChannelNumber} {FileMode}: {FileModeOption}] {driver} {file/device}
```

DIRECT_INPUT

To open a file or device for sequential input.

```
Direct_Input [Channel {ChannelNumber} {FileMode}: {FileModeOption}] {driver} {file/device}
```

APPEND_OUTPUT

To add output to a sequential file, a printer, or the Windows clipboard.

```
Append_Output [Channel {ChannelNumber} {FileMode}: {FileModeOption}] {driver} {file/device}
```

CLOSE_INPUT

To close a sequential input file.

```
Close_Input [Channel {channel-no}]
```

CLOSE_OUTPUT

To close a sequential output file.

```
Close_Output [Channel {channel-no}]
```

General Purposes

ERROR

To signal an error has occurred.

```
Error {error-num} [{message}]
```

ABORT

To end the execution of the current DataFlex program. Implemented for Windows only. If used, confirmation to exit application will be provided.

```
Abort
```

SLEEP

To pause execution of the DataFlex program for a specified time interval.

```
Sleep seconds
```

Supported Compiler Directives

USE

To incorporate separate source files into DataFlex programs at compile time as source code. Use searches for file {SourceFileName.ext} at the time the program is compiled. If the specified file is found and this file has not already been included in the program, its contents are loaded compiled as though they were part of the program.

```
Use ["] [{path}]{SourceFileName}[{.ext}]} ["]
```

#REM

To generate a comment at compile time.

```
#Rem path\filename.ext
```

#ERROR

The #Error directive declares a compile time error.

```
#Error {error-number} {error-message}
```

#INCLUDE

To incorporate into a program at compile time, program source code from another source-code file. The entire file filename.ext will be included in the compilation process at the point(s) of the #Include(s).

```
#Include path\filename.ext
```

Parser

DataFlex Scriptor parser accepts CRLF terminated line of the characters (bytes) and going from the left to the right splits it to the separate tokens.

Parser tokens are abstract and identify basic language element types (Symbols).

At the next (pre-compiler) stage, these tokens will be identified and appropriate actual symbol IDs allocated.

Parser can recognize the following basic tokens:

1. End of File
2. General Error (token cannot be recognized)
3. End of Line
4. Code comment
5. String, Real, Number and Integer constant
6. Abstract command
7. Element ID (variable, constant, object name etc.)
8. Operator (+, -, *, / etc.)
9. Expression start and stop ('(', ')')
10. Native array definition ('[', ']')
11. Tag start and stop ('{', '}')
12. List of the parameters (',')
13. Compiler directive (#INCLUDE, #ERROR etc.)
14. Database field (FILE.FIELD)
15. DataFlex Scriptor Scope name

Pre-Compiler

The purpose of the pre-compiler is to prepare parsed script for execution.

The main actions of compiler:

1. Collect all included source files into the single script.
2. Collect tokens and create script scopes.
3. Define and build list of variables for the each scope defined.
4. Recognize DataFlex phrases and allocates appropriate Symbol IDs for all tokens.
5. Control script consistency and validate phrases (reporting compile time errors).
6. Replace the token names with their respective script element IDs and references.
7. Remove noise lines[1].

When compilation process completed, all script tables are built and ready to be executed.

[1] Noise lines are lines like Tags ({}), Variable declarations (Since they are registered), Include file commands (USE/#INCLUDE), Open command (since file declared and opened), END* scope commands (which identify end of scope), scope start commands (since scope is started) etc etc. So roughly speaking, any lines that are not required for the script execution purposes

Builder

Option to build script and generate "executable" (*.DE) file is introduced to improve performance and secure script code in case of deployment.

Executable script may be loaded into the DataFlex scriptor and executed without compilation.

Build process reads all collected tables (scopes and variables) and records them into the binary file.

The structure of the binary file as following:

1. File Header Record:

```
[VERSION][INFO(File, Date, Time, Copyright)]
```

2. Script scopes and files record:

```
[FilesCount][FileTable: MDDLLLLNNFFFF*, ...]
```

3. Scopes record:

```
[ScopeCount][ScopeHeader:TTAARRDDLLNNNN**][SkopeTokens:[TokensCount][TokensTable: SSLLLLCCFFBBBBIIII*, ...]]
```

4. Variables record:

```
[VariablesCount][VariablesTable: TTLLLLCCDDBBNNNN*BBBBVVVV*, ...]
```

File/Scope Name Record

MDDLLLLNNFFFF*, where

Element	Description	Type
M	Method (Script, USE...)	BYTE
DD	Depth Level	SHORT, 2
LLLL	Line No	DWORD, 4
NN	Size of File Name	SHORT, 2
FFFF*	File/Scope Name	CHAR, Variable

Scope Record

TTAARRDDLLNNNN*, where

Element	Description	Type
TT	Scope Type (Function,..)	SHORT, 2
AA	No. Arguments	SHORT, 2
RR	Returns Type	SHORT, 2
DD	Return Type Dimension	SHORT, 2
LL	Size of Name	SHORT, 2
NNNN*	Scope Name	CHAR, Variable

Scope Tokens Record

SSLLLLCCFFBBBBIIII*, where

Element	Description	Type
SS	Symbol	SHORT, 2
LLLL	Line No	DWORD, 4
CC	Column No	SHORT, 2
FF	File Index, See File Table Record Ind	SHORT, 2
BBBB	Size of Image	DWORD, 4
IIII*	Image	CHAR, Variable

Variable/Constant List Record

TTLLLLCCDDSSBBNNNN*BBBBVVVV*, where

Element	Description	Type
TT	Type	SHORT, 2
LLLL	Line No	DWORD, 4
CC	Column No	SHORT, 2
DD	Dimension	SHORT, 2
SS	Scope Ref.	SHORT, 2
BB	Size of Name	SHORT, 2
NNNN*	Name	CHAR, Variable
BBBB	Size of Value	DWORD, 4
VVVV*	Value	CHAR, Variable

Security Options

Options below are incorporated to provide minimal level of the security on scripting. All these options are enabled by default and you may switch them OFF on DFScriptor class instantiation. DataFlex Scriptor will protect this options to disallow change from the custom script.

```
Class oDFScriptor1 is a cDFScriptor
    Set pbAllowDFScriptSave          To False
    Set pbAllowDFScriptDelete        To False
    Set pbAllowDFScriptRestructure   To False
    Set pbAllowDFScriptTransaction   To False
    Set pbAllowDFScriptFind          To False
    Set pbAllowDFScriptAttributes    To False
End_Object
```

Options Description

Option (Property)	Description
pbAllowDFScriptSave	Cancels DataFlex *SAVE* and *SAVERECORD* commands execution.
pbAllowDFScriptDelete	Cancels DataFlex *DELETE* command execution.
pbAllowDFScriptRestructure	Cancels DataFlex *STRUCTURE_START* and *STRUCTURE_END* commands execution.
pbAllowDFScriptTransaction	Cancels DataFlex *BEGIN_TRANSACTION* and *END_TRANSACTION* commands execution.
pbAllowDFScriptFind	Cancels DataFlex *FIND* command execution.
pbAllowDFScriptAttributes	Cancels DataFlex SET_ATTRIBUTE command execution.

To avoid provided security, you should create respective methods in your application, provide application security level and call these methods from the script, rather using it directly.

DataFlex Scriptor High Level Methods

Defining Scriptor Instance

To create DataFlex Scriptor Object in the Application, you must instantiate new Scriptor object from cDFScriptor class:

```
Use dfScriptor.pkg

Object oDFScriptor1 is a cDFScriptor
End_Object
```

You may create as much script objects as required or use single instance in all cases, re-loading custom scripts each time. Alternatively, you may use Script scopes in the single script and call scope rather than main script.

Loading Script

First of all, custom script should be loaded into the Scriptor object. You may use both - source files (.DS) or executable (.DE) files. Executable files should improve performance and security in case of deployment.

Loading Script Source

To load script source, use property *psScriptSource* and method *Compile*. To initialize object, use *Delete_Data* method:

```
// Custom Application Code
Procedure mScriptOpen
    Send Delete_Data    to oDFScriptor1
    Set  psScriptSource of oDFScriptor1 To "MyScript.ds"
    Send Compile        to oDFScriptor1
    // Send DFS_Exec      to oDFScriptor1    ""
End_Procedure
```

Loading Script Executable

To load pre-compiled (executable) script file, use property *psScriptSource* and method *Compile* or *Load_Executable*. To initialize object, use *Delete_Data* method as well:

```
// Custom Application Code
Procedure mScriptExeOpen
    Send Delete_Data    to oDFScriptor1
    Set  psScriptSource of oDFScriptor1 To "MyScript.de"
    Send Compile        to oDFScriptor1
End_Procedure
```

or

```
// Custom Application Code
Procedure mScriptExeOpen
    Send Delete_Data     to oDFScriptor1
    Set  psScriptSource  of oDFScriptor1 To "MyScript.de"
    Send Load_Executable to oDFScriptor1 (psScriptSource(Self))
End_Procedure
```

Calling Script

When script loaded and compiled (or excutable script is loaded), you man run it. To call script from the application (binding script to some GUI or Web component), EXEC_SCRIPT method should be used:

```
// Custom Application Code
Procedure mScriptRun
    // This will run Main Script
    Send DFS_Exec to oDFScriptor1 ""
End_Procedure
```

or

```
// Custom Application Code
Procedure mScriptRun
    // This will run mSomeScriptMethod
    Send DFS_Exec to oDFScriptor1 "mSomeScriptMethod"
End_Procedure
```

Passing Parameters

If you need to pass some parameters into the script, you should use psScriptParameters property (Defined as String[]):

```
// Custom Application Code
Procedure mScriptRun
    String[] sScriptParameters
    Move 1 To sScriptParameters[0]
    Move 2 To sScriptParameters[1]
    // This will run mSomeScriptMethod with 2 parameters
    Set psScriptParameters to sScriptParameters
    Send DFS_Exec to oDFScriptor1 "mSomeScriptMethod"
End_Procedure
```

Obviously, Main script or its scope must define parameters to be passed. If parameters are not defined, but passed, they will be ignored. If passed parameters more than defined, then will be accepted only defined parameters and others are ignored. If you passed less parameters, then error will be generated " Required script argument missed".

Returning Script Value

If you need to return script value, then use "Function_Return" or "Procedure_Return" commands for the script scope and "Script_Return" for the main script. Please note, "Script_Return" is not DataFlex command, so you you will try compile your script by DataFlex Compiler, more likely you will get "Unknown command" error. This will be fixed in the later versions of DFScriptor.

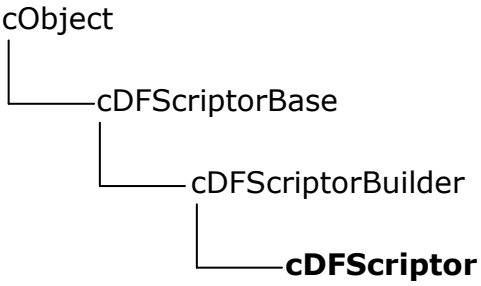
When DFScriptor will find one of the above commands, it will stop execution and will return value specified using *psReturnValue* property, which you can then read and use as required:

```
// Custom Application Code
Function mScriptRun String sName Returns String
    String sRetValue
    // This will run Main Script (Script_Return command expected)
    Send DFS_Exec      to oDFScriptor1 ""
    Get  psReturnValue of oDFScriptor1 To sRetValue
    Function_Return sRetValue
End_Procedure
```

Class Description

Class cDFScriptor is used to instantiate DataFlex Scriptor object and provides all the futures to create and run scripts in the custom applications.

cDFScriptor Hierarhy



cDFScriptor Properties

Property	Type	Description
ptTokens	Private	Internal storage of the parsed tokens.
ptVariables	Private	Internal storage of the Script Variables.
ptDFSFiles	Private	Internal storage of the Script general structure.
ptCompilerInfo	Private	Internal storage of the Compiler messages.
ptDFSErrors	Private	Internal storage of the compilation errors.
ptDFSScopes	Private	Internal storage of the script scopes.
psScriptParameters	Public	Used to pass parameters into the script or script scope.
psReturnValue	Public	Used to return script or scope values.
piGeneratePRN	Public	If enabled, PRN file will be generated (PRN may be switched OFF).
piPRNChannel	Public	Channel to be used during compilation to generate PRN and TBL files.
piScriptChannel	Public	Channel to be used during compilation to read script source files.
psScriptSource	Public	Script file name including full path.
psScriptFile	Public	Script file title (short name).
piExecutable	Private	Internal. Identifies either loaded script is executable (.DE).
ptsCompileTime	Private	Internal. Calculates compile time (TimeSpan).
piErrorsCount	Public	Read only. Compilation errors count.
piLinesCount	Public	Read only. Counts overall number of script lines during compilation.
piTokensCount	Public	Read only. Counts overall number of script tokens during compilation.

Property	Type	Description
piCommandsCount	Public	Read only. Counts overall number of script commands during compilation.
piDirectivesCount	Public	Read only. Counts overall number of script directives during compilation.
piVariablesCount	Public	Read only. Counts overall number of script variables during compilation.
piConstantsCount	Public	Read only. Counts overall number of script constants during compilation.
piObjectsCount	Public	Read only. Counts overall number of script objects during compilation.
piMethodsCount	Public	Read only. Counts overall number of script methods during compilation.
piScopeStartCount	Private	Provides control during compilation on Scope start and end.
piScopeStopCount	Private	Provides control during compilation on Scope start and end.
pbAllowDFScriptSave	Public	Security option. Allow SAVE and SAVERECORD commands.
pbAllowDFScriptDelete	Public	Security option. Allow DELETE command.
pbAllowDFScriptTransaction	Public	Security option. Allow BEGIN_ and END_ TRANSACTION commands.
pbAllowDFScriptFind	Public	Security option. Allow FIND command.
pbAllowDFScriptRestructure	Public	Security option. Allow STRUCTURE_START and STRUCTURE_END commands.
pbAllowDFScriptAttributes	Public	Security option. Allow SET_ATTRIBUTE command.
psName	Private	Internal. Used to prevent DFScriptor object reference from the script.

cDFScriptor Methods

Method	Type	Description
sDFScriptorBase		
Add_Method_Info	Procedure	Adds information about new method into the methods list.
Add_Error_Info	Procedure	Registers new compile error found.
Add_Compiler_Info	Procedure	Adds new message into the Compiler Information list to be shown if required.
Clear_Script	Procedure	Procedure clears current script EXEC info and makes it ready to run script again.
Delete_Data	Procedure	Procedure deletes all cuurent script data.
Scope_Name	Function	Returns Scope name by ID.
Command_ID	Function	Returns command ID by token image or DFT_ERROR.

Method	Type	Description
Scope_Type_Description	Function	Returns available scope types (Class, Object, Methods) or Unknown.
Keyword_ID	Function	Returns keyword or command ID by token Image.
Var_Type	Function	Returns Variable type by Command ID provided.
Var_Type_String	Function	Returns type name by Variable Type ID provided (Debug and Output purposes only).
Var_Prefix	Function	Returns variable prefix: "|FI"=File item, "|CI"=Constant item, "|VI"=Variable item
fDFSOutDim	Function	Returns Dimension Image (Debug and Output purposes only).
PrintCharsets	Procedure	Prints list of charsets available (Output tables purpose)
PrintSymbols	Procedure	Prints list of symbols available (Output tables purpose)
PrintTokens	Procedure	Prints list of tokens available (Output tables purpose)
PrintVariables	Procedure	Prints list of variables available (Output tables purpose)
DebuggerInfo	Procedure	Prints all lists available (Output tables purpose)
Save_Executable	Procedure	Builds script executable file (*.DE).
Load_Executable	Procedure	Reads script executable file (.DE) and loads data
Object_Short_Name	Function	Returns short object name.
cDFScriptoBuilder		
Token	Set	Adds new single token into the list of tokens
Token	Function	Returns token image by token index
Process_Compiler_Expression	Function	Evaluates compile-time expressions
Init_Constant	Procedure	Initializes constant value at the point of defining (Define C_ for "Value")
Var_Value	Set	Assigns value to the variable (Variable should be passed as an Index with prefix, eg. |VI23)
Var_Value	Function	Returns value of the variable (Variable should be passed as an Index with prefix, eg. |VI23)
CompareVarInternal	Function	Internal. Function to properly allocate Global and Local variables for the scopes
IsDatabaseFile	Function	Returns true if passed image is a database file reference.
Var_Register	Function	Adds new variable into the storage of variables
Var_Find	Function	Finds variable by image and returns prefixed variable index (ge. "iTest" >> "|VI73")
Process_Variable	Function	Defines new variable

Method	Type	Description
Process_Constant	Function	Defines new constant and initializes the value of necessary
Type_Dimension	Function	Returns current type dimension (eg. "aiTest[4][1]"=2, "sNames[]"=1 etc.)
Register_Variables	Procedure	Declares Variables/Constants
IsFileField	Function	Returns TRUE if passed token is a database file/field token.
Validate_ID	Function	Validates ID token (DFT_ID) if not identified, returns 0
RegisterTokens	Procedure	Adds parsed line tokens into the processed scope tokens list
AddLineToken	Procedure	Helper procedure to collect line tokens
ParseLine	Procedure	Main parsing procedure. Parses script line and builds line tokens list.
mIncludeFile	Function	Function to compile current source file (.DS). Called recursively to include used/included source files (.DP). Also, in line processed some compiler directives.
Compile	Procedure	Script compile procedure. Requires script file. Used by debugger to collect compile information.
	cDFScriptor	
Return_Statement	Function	Helper function to return script or current scope value.
Scope_Value	Function	Executes current script scope in case called as a function in expression or using Set/Get/Send commands.
Process_Expression	Function	Executes and returns run-time expression value.
Skip_Tokens	Function	Skips unnecessary tokens from the start and to the stop symbol specified.
Process_Block	Function	Executes block of the code between Start and Stop tokens
SetFieldValue	Procedure	Set database field value.
GetFieldValue	Function	Returns current database field value.
GetScriptValue	Function	Returns current token(s) script value - script or DataFlex constants, File/Field, Script ID or Expression.
Sleep_Statement	Function	SLEEP command processor.
SysDate_Statement	Function	SYSDATE command processor.
Indicate_Statement	Function	INDICATE command processor (Pre-defined indicators only).
Move_Statement	Function	MOVE, ADD, SUTRACT, INCREMENT, DECREMENT command processor.
GetSetAttr_Statement	Function	GET_ATTRIBUTE/SET_ATTRIBUTE commands processor.

Method	Type	Description
InOutput_Statement	Function	Input/Output commands processor (DIRECT_OUTPUT, CLOSE_INPUT, READ*, WRITE* etc).
GetSetSend_Statement	Function	GET, SET, SEND commands processor.
dbFile_Statement	Function	Database related commands (OPEN, CLEAR, SAVE etc.) processor.
Find_Statement	Function	FIND command processor.
DBTrans_Statement	Function	BEGIN_TRANSACTION/ END_TRANSACTION commands processor.
DBStruct_Statement	Function	STRUCTURE_* commands processor.
For_Statement	Function	FOR/LOOP scope processor.
Error_Statement	Function	ERROR command processor.
Repeat_Statement	Function	REPEAT/UNTIL scope processor.
While_Statement	Function	WHILE/LOOP scope processor.
Process_Else	Function	Internal. Processes ELSE command for the IF.
IfElse_Statement	Function	IF/ELSE commands/scopes processor.
Exec_Command	Function	Scope commands processor.
Exec_Script	Function	Script Scope processor by Scope ID.
DFS_Exec	Procedure	Executes script by name. If name is not specified, DFS_DESKTOP scope is executed. Otherwise scope by name (Procedure or Function)

Expanding Possibilities

Adding New Command

This is step-by-step instruction on how to add new command support for the DFScriptor.

1. Define new command under dfScriptor.h file, see "DFScriptor Symbols" enumeration list. Make sure you add command into the right place and provide it with DFC_ prefix. Make sure you do not break command ranges like DFC_WHILE/DFC_CLASS, DFC_LOOP/DFC_END_CLASS, DFC_MOVE$START/DFC_MOVE$STOP or workout them correctly (check RegisterTokens procedure).

2. Add new command into the same place of the dfsStaticLoadSymbols procedure (dfScriptor.h). It allows to print correct symbols table and will simplify debug purposes.

3. Under file dfScriptor.pkg find Command_ID function and add new code to recognize your new command. Please note - parser will uppercase command token automatically. So, add image name and return your new DFC_ constant.

4. Under file dfScriptor.pkg find and amend Exec_Command function, adding processor method for your new command. Please note - processor must accept current token and return last token after processing. If you have implement set of the similar command, you may provide single handler. In this case, you may pass current token Symbol to recognize command found.

5. Create new command Handler function. To keep consistency, name it as <NEWCMD>_Statement function.

6. Compile and run sample application, create new or open existing script, add new command statement and make sure it compiled and processed correctly.

Extending Functionality

You may add any functionality to the Scriptor by creating new methods, classes and objects in the your custom application. Scriptor support most of general DataFlex data access and processing commands. So rather editing Scriptor source, just create new cool functionality in application code and simply use it in the script. See DFSSystem.pkg for example and sample script package files (*.DP). This sample creates number of useful constants, which can be used in your custom script.

Under DataFlex Script Debugger, Variables page you may see values of the pre-defined constants.

Project File Reference

File	Description
dfScriptor.pkg	Source code of the DataFlex Scriptor Class
dfScriptor.h	DataFlex Scriptor header file. It includes all general structures, constants and static methods to support Scriptor class.
DFSSystem.pkg	Defines number of useful constants and methods, that can be used in the custom scripts (DataFlex attributes, Workspace paths, Date and Time functions etc.).
DFSDebugger.vw	DataFlex Scriptor debugger view. This is optional component and not required for the deployment.
dfsScriptEditor.pkg	Source code for the source editor based on the Codemax control.
cCodeMax.h	Codemax control support file.
cCodeMax.pkg	TheHamme Codemax control implementation.
cWinFunc.pkg	Codemax control support file.
mPointer.pkg	Codemax control support file.
mStrConv.pkg	Codemax control support file.
mWinAPIGetKeyNameText.pkg	Codemax control support file.
cWinImageList.pkg	DFS Debugger and Codemax control support file.
cWinListView.h	ListView control support file.
cWinListView.pkg	ListView control implementation.
drgndrop.pkg	DFS Debugger Support File.
ScriptorAbout.pkg	Sample DFS Project About Dialog.
dfsConsole.pkg	DFS Scriptor sample - DataFlex Debugger Console. Available under Debugger tab page.
system.dp	System DataFlex Scriptor Package. This file defines set of the predefined constants to simplify script writing.
sysattr.dp	Included into the System.dp
syslocale.dp	Included into the Sysattr.dp
sysws.dp	Included into the System.dp
***.ds**	Scriptor sample sources
***.dp**	Scriptor sample packages
***.de**	Scriptor executables
***.prn**	Scriptor PRN files (generated on compile)
***.tbl**	Scriptor TBL files (generated on compile)

General Feedback

You may send your bug reports as well as suggestions or fixes to the following addresses:

Sergey V. Natarov

senatc@postman.ru, senatc@mail.ru

DAW Support Forum

http://support.dataaccess.com/Forums/showthread.php?60391-DataFlex-Scriptor-2-(Alpha-1)

GIT Hub

https://github.com/senatc/DFScriptor