# Blitzprog SDK

## *Komponenten*

- API - Application Programming Interface
- IDE - Integrated Development Environment
- MDA - Model Driven Architecture

## *Details*

- Standard-Bibliothek
- Klassenkapselung
- Parser
- Interne XML-Struktur

## *Blitzprog API*

### *Beschreibung*

Die Blitzprog API ist der Kern des Software Development Kits. Sie besteht aus dem in C++ geschriebenen Code und ist die komplexeste Komponente, die auf der höchsten Hierarchieebene steht. Wenn die API verändert wird, müssen alle Programme, die darauf basieren, angepasst bzw. umgeschrieben werden.

### *Ziel*

Eine einfache, effiziente und benutzerfreundliche Programmierschnittstelle zu entwickeln.

### *Features*

- Modulverwaltung
- Multithreading (parallele Ausführung von Funktionen)
- Thread-Sicherheit (thread-safe)
- 64-Bit-Unterstützung
- Portabilität (Linux und Windows)
- Ausnahmebehandlung (exception handling)
- Speichermanagement
- Grafische Benutzeroberflächen (GUI)
- Effiziente Grafikfunktionen (2D und 3D, inkl. Partikelsystem)
- Netzwerk (TCP/IP, UDP, IPv6 und IPv4)
- Mathematik (Winkelfunktionen, 2D/3D-Vektoren, 4x4-Matrizen)
- Dateifunktionen
- Datums- und Zeitfunktionen
- Eingabegeräte: Maus, Tastatur und Joystick

- Pixmaps für GUIs und Images bzw. Textures für Grafik
- Einheitliches Stream-Interface für Filestreams und Networkstreams
- Datenstrukturen (Arrays, Listen, Maps, usw.)
- Unicode und ASCII

## *Blitzprog IDE*

### *Beschreibung*

Die Blitzprog IDE ist die Entwicklungsumgebung für die oben genannte API.

### *Ziel*

Eine nicht überladene, einfach zu bedienende und auf der Blitzprog API basierende Entwicklungsumgebung bereitzustellen.

### *Features*

- Code-Editor
- Syntaxüberprüfung während der Codierungsphase
- Externer Compiler (für externe Tools)
- Profiler (misst die Zeit, die die jeweiligen Funktionen beim Testvorgang gebraucht haben)
- Debugger
- Syntax Switcher (Syntax kann während der Codierung verändert werden)
- XML-Vorschau (siehe Blitzprog MDA)
- Auto-Complete mit künstlicher Intelligenz
- Syntaxhighlighting
- Projektverwaltung
- Projektvorlagen
- Auto-Correct (Korrektur bei bekannten Keywords/Befehlen)
- Auto-Indent (automatisches Einrücken, auch beim Einfügen von Codes aus der Zwischenablage)
- Code-Folding
- Gleichzeitiges Bearbeiten von Codes mit anderen Personen über das Internet
- Klassendesigner
- Codeschnipsel für häufig verwendete Codes (z.B. kleine mathematische Funktionen)
- Automatisch generierte ToDo-Liste
- Drag 'n' Drop
- Portabilität (Linux und Windows)

## *Blitzprog MDA*

### *Beschreibung*

Blitzprog MDA ist ein Programm, welches Code aus einem Modell ableiten kann. Das Modell wird dabei universell mit XML beschrieben.

### *Ziel*

Code sollte nicht länger in einfachen Textdateien gespeichert werden. Anstatt herstellerspezifische Erweiterungen wie z.B. das Auslesen von ToDo-Einträgen aus Kommentaren oder die Metainformationen für javadoc zu verwenden, sollte man ein einheitliches Modell - basierend auf XML - entwickeln und von diesem Modell Code ableiten können.

### *Features*

- Man kann Code für die wichtigsten Programmiersprachen erzeugen, ohne alles neu schreiben zu müssen.
- Die Formatierung des Codes/Modells spielt keine Rolle (siehe XML), beim abgeleiteten Code können mehrere Ausgabeoptionen eingestellt werden
- In den Codedateien können Header und Metadaten untergebracht werden (Autor, Website, E-Mail, Version, Beschreibung, Linker- und Compilereinstellungen, usw.)



# Standard-Bibliothek

## *Modulverwaltung*

### *Beschreibung*

Eine Modulverwaltung erleichtert das Einbinden von Modulen. Im einfachsten Fall muss man nur das Modul in den Modul-Ordner kopieren und man kann es ohne Probleme importieren.

### *Ziel*

Ein benutzerfreundliches und einfach zu verwendendes Modulsystem zu entwickeln, bei dem objektorientierte Programmierung eine herausragende Rolle spielt.

### *Features*

- Einfaches Importieren von Modulen
- Jedes Modul erhält automatisch einen Namespace
- Module von der Standard-Bibliothek können direkt ohne Angabe des Namesraums importiert werden, es reicht also ein Import Math im Gegensatz zu Import Blitzprog.Math
- Theoretisch sind beliebig viele Hierarchieebenen möglich, z.B. Blitzprog.Math.Vector.3D, dadurch kann man zusammenhängende Module besser kapseln

## Multithreading

### Beschreibung

Multicoreprozessoren haben sich durchgesetzt und bieten durchschnittlich 30 % mehr Performance, wenn die Anwendung auf mehrere Prozessorkerne optimiert worden ist (im besten Fall sind 100 % möglich, aber das wird nur sehr selten erreicht).

### Ziel

Eine Thread-API zu entwickeln, die sowohl über Thread-Objekte mit Funktionszeigern genutzt werden kann als auch mit eigenen Klassen und einer Threadbasisklasse, so wie es in Java der Fall ist.

### Features

- Die gesamte Standardbibliothek wird Thread-Safe sein.
- Globale Variablen werden automatisch als volatile angesehen (Ausnahme: Die WPA zeigt, dass keine vom Programmierer erstellte Threads verwendet werden)

## 64-Bit-Unterstützung

### Beschreibung

Der Datentyp Size , der in der Blitzprog API häufig benutzt werden soll, ist auf 64-Bit-System 64 Bit groß und auf 32-Bit-Systemen 32 Bit. Wenn eine nicht-negative Zahl benötigt wird, die bspw. eine Größe darstellt, dann sollte immer der Typ Size verwendet werden (entspricht size_t in C++).

### Ziel

Es sollte auf 64-Bit-Systemen keine Probleme beim Kompilieren geben.

### Features

- Datentyp Size zum Darstellen von Größenangaben, zum Beispiel für die Länge eines Arrays.

## Portabilität

### Beschreibung

100-prozentige Portabilität ist nur gegeben, wenn das Programm ohne jegliche Änderungen auf dem Zielsystem ausgeführt werden kann. Eine ca. 99,9-prozentige Portabilität ist gegeben, wenn man den Code nicht ändern, sondern nur neu kompilieren muss.

### Ziel

Eine fast vollständige Portabilität. Man muss den Code nur rekompilieren.

### Features

- Bedingte Code-Kompilierung: Es wird für das jeweilige Zielsystem immer die effizienteste Methode gewählt.

## Ausnahmebehandlung

### Beschreibung

Ausnahmebehandlung erhöht die Sicherheit von Programmen und bietet ein modernes Fehler-Abfangsystem .

### Ziel

Die Sicherheit von Anwendungen zu erhöhen und ständige Abfragen wie if(errorCode == -1) zu entfernen.

### Features

- Try&Catch-Ausnahmebehandlung wie in C++
- Exit-Codes werden über die End(int exitCode) Funktion übergeben, die nichts anderes tut als throw exitCode; auszuführen, dies wird in einem Top-Level-Try/Catch-Block der main-Funktion abgefangen.

## Speichermanagement

### Beschreibung

Das Speichermanagement in der Blitzprog API übernehmen Smart Pointer. Smart Pointer besitzen einen Referenzzähler, der atomar inkrementiert bzw. dekrementiert wird, wenn ein weiterer Smart Pointer auf dasselbe Objekt zeigt bzw. dieses Objekt nicht mehr benötigt wird. Es gibt 2 Ansätze, um Smart Pointer (im Folgenden SharedPtr genannt) zu implementieren:

1. Klasseninstanzen/Objekte werden direkt über SharedPtr-Templates deklariert.
2. Alle Klassen erben die Eigenschaften/Methoden von SharedPtr.

Die erste Methode wird von Boost verwendet. Sie hat allerdings einen Nachteil: Der this-Zeiger bleibt weiterhin ein Zeiger auf das Objekt selbst, nicht auf den SharedPtr des Objekts. Beispiel:

```
XMLNode node = New XMLNode(„node", parent);
```

XMLNode ist hier ein Synonym für SharedPtr<TXMLNode>:

```
typedef SharedPtr<TXMLNode> XMLNode;
```

Der Konstruktor könnte wie folgt implementiert sein:

```
TXMLNode(const String &nName, SharedPtr<TXMLNode> parent) //name...
{
    if(parent)
    {
        parent->Add(this);
        this->parentWeakPtr = parent;
    }
    ...
}
```

Hier wird *this* an die Methode Add als Parameter übergeben.Der Referenzzähler wird nicht erhöht

und es könnte zu Problemen kommen, im schlimmsten Fall stürzt das Programm ab, obwohl der Programmierer eigentlich sauberen Code geschrieben hat.

Die zweite Methode umgeht dieses Problem, indem jede Klasse alle Eigenschaften/Methoden von SharedPtr erbt. Die this-Zeiger sind somit Zeiger auf die SharedPtr selbst. Jedes Objekt besitzt einen eigenen Referenzzähler. Dies ist übrigens auch die einzige Möglichkeit, wenn man Operatorenüberladung nutzen will. Mit der ersten Methode wäre dies nicht möglich, weil alle Variablen SharedPtr sind.

#### *Ziel*

Komfort und Sicherheit zu vereinen, indem jedes Objekt sich automatisch selbst entfernt, wenn es nicht mehr benötigt wird.

#### *Features*

- Man muss sich nicht selber um die Verwaltung des Speichers kümmern.
- Das Programm wird robuster, weil ein Objekt erst freigegeben wird, wenn es nicht mehr benötigt wird und weil bei jedem Zugriff vorher überprüft wird, ob es sich um einen Null-Zeiger handelt.
- Ein relativ geringer Overhead, den man in Kauf nehmen kann, denn SharedPtr sind besonders für Threads nützlich und bieten eine hohe Sicherheit, da sie ebenfalls Thread-Safe sind.

## *Grafische Benutzeroberflächen (GUI)*

#### *Beschreibung*

Die grafische Benutzeroberfläche ist eine der wichtigsten Komponenten. Sie muss portabel und ans Betriebssystem anpassbar sein. Ebenso spielt das Ereignis-System eine große Rolle. Die Blitzprog GUI greift auf die Standard-Widgets des Betriebssystem zu und nutzt Callbacks.

#### *Ziel*

Eine möglichst einfache Schnittstelle für grafische Oberflächen zu entwickeln, die auf jedem Zielsystem anders implementiert werden kann.

#### *Features*

- Alle elementaren Komponenten einer GUI sind hier ebenfalls vorhanden.
- Callbacks mit einfacher Parameterübergabe, z.B. button->OnClick(ShowMsg,  Hello World  );
- Eine auf AJAX basierende Implementation der GUI, damit man reine GUI-Programme auch im Browser ausführen kann.

## *Effiziente Grafikfunktionen*

#### *Beschreibung*

Die wichtigsten Grafikfunktionen für 2D und hardware-beschleunigtes Single-Surface-Rendering

sind das Minimum bei dieser Grafik-API für OpenGL/DirectX. Ob eine 3D-Komponte realisiert wird, steht zu diesem Zeitpunkt noch nicht fest.

### *Ziel*

Schnellere 2D-Funktionen als in Blitz3D und Max2D.

### *Features*

- Gepuffertes Single-Surface-Rendering mit Front-To-Back-Sortierung und korrektem Alpha-Blending
- Laden von BMP-, JPG- und PNG-Dateien, weitere Dateiformate können theoretisch über die abstrakte Schnittstelle implementiert werden

## *Netzwerk*

### *Beschreibung*

Siehe BNetEx.

### *Ziel*

Eine Low-Level-API mit zusätzlichen High-Level-Funktionen zu realisieren.

### *Features*

- TCPStream
- UDPStream
- IPv6, IPv4
- Pipelining, wenn möglich.
- Thread-Safe

## *Mathematik*

### *Beschreibung*

Ohne Mathematikfunktionen kann eine Grafik-API nicht funktionieren, deswegen gibt es ein Mathematik-Modul, bei dem Funktionen wie Sin, Cos aber auch Klassen wie Vector3D und Matrix4x4 implementiert sind.

### *Ziel*

Grundlegende Mathematikfunktionen sowie Klassen/Funktionen für lineare Algebra bereitzustellen.

### *Features*

- Vector3D
- Matrix4x4
- Sin, Cos, Tan, SinRad, CosRad, TanRad
- ASin, ACos, ATan, ASinRad, ACosRad, ATanRad

- Abs, Sgn, usw.

## *Dateifunktionen*

### *Beschreibung*

Zugriffe auf das Dateisystem sollten möglichst effizient, aber dennoch komfortabel (z.B. ReadLine) sein.

### *Ziel*

Eine auf den C++ IOStreams basierende FileStream-API zu realisieren.

### *Features*

- Read-Funktionen, Write-Funktionen, auch zum Einlesen von Zeilen
- ReadBytes und WriteBytes zum Lesen bzw. Schreiben von Datenblöcken
- FileStream schließt sich automatisch, wenn er nicht mehr gebraucht wird, eine Close-Funktion ist dennoch vorhanden

# Klassenkapselung

- Stream
  - FileWriteStream
  - FileReadStream
  - TCPStream
  - UDPStream
- GraphicsDriver
  - OpenGL
  - Direct3D9
- Audio
  - OpenAL
  - WAVLoader
  - OGGLoader
- Collection
  - Array
  - List
  - Map
- DateTime
  - Date
  - Duration
- Math
  - Vector2D

    - Vector3D
    - Matrix4x4
    - Rect
- XML
    - XMLFile
    - XMLNode
- INI
    - INIFile
    - INICategory
- GUI
    - Widget
    - Window
    - Panel
    - TabContainer
    - Tab
    - Button
    - CheckBox
    - RadioButton
    - TextField
    - TextArea
    - ListBox
    - TreeView
    - ComboBox
    - HTMLView
    - ProgressBar
    - ToolBar
    - GUIFont
    - Menu
    - IconStrip
    - Icon
- Input
    - Keyboard
    - Mouse
    - Joystick
- Network
    - TCPStream
        - TCPServer

        - HTTPStream
    - UDPStream
        - UDPServer
    - IP
- Pixmap
    - BMPLoader
    - PNGLoader
    - JPGLoader
    - TGALoader
- String
- Thread

# Parser

### *Beschreibung*

Der Parser von Blitzprog MDA ist eine ausführbare Datei, die man mit der Kommandozeile oder einem grafischen Front-End wie z.B der Blitzprog IDE bedienen muss.

### *Optionen*

- -o

Spezifiziert die Ausgabedatei bzw. die Ausgabesprache. Mögliche Parameter sind: File.cpp, Pfad/Datei.cpp, C++, cpp, usw.

- -d

Erzeugt eine Dokumentation aus dem Code. Als Parameter kann eine Ausgabe-HTML-Datei angegeben werden.

- -s

Fügt zur Dokumentation ebenfalls Codestatistiken hinzu.

- -p

Erzeugt Profilerinformationen für den Code. Die Zeit, die jede Funktion benötigt, wird gemessen und beim Beenden des Programms in eine HTML-Datei geschrieben. Siehe Blitzprog Profiler.

### *Performance*

Das Parsen einer BPC-Datei und das Umwandeln bzw. Speichern in XML-Code dauert ca. 12 ms.

Eine Test-Tabelle mit mehreren Läufen :

| | |
|---|---|
| 1. | 12 ms |
| 2. | 5 ms |
| 3. | 4 ms |
| 4. | 3 ms |
| 5. | 2 ms |

Nach dem 5. Übersetzungsvorgang derselben Datei sind es konstant 2-4 ms, weil die Festplatte anscheinend einen guten Cache hat. Das Umwandeln wurde mit der unvollständigen Parser-Version 0.1.3 und einer 76 Zeilen langen BPC-Datei getestet. Je größer die Datei ist, umso effizienter arbeitet der Parser, weil das Öffnen einer Datei relativ langsam ist.

### *Features*

- Hohe Performance
- GCC-ähnliche Options-Syntax
- Dokumentationserzeugung direkt bei der Kompilierung möglich, wenn erwünscht.
- Integrierter Profiler. Da er auf XML basiert, ist die Ausgangssprache irrelevant.

## © Eduard Urbach - 01. November 2007

## Interne XML-Struktur

### *Variablen-/Parameterdefinition*

```
<var name="[string]" const="[bool]">
    <type callby="reference|value" pointer="[int]">
        [...]
    </type>
    <value>
        [...]
    </value>
</var>
```

***Hinweis:*** *Die Eigenschaften callby und pointer werden nur in der Standardbibliothek verwendet. In normalen Codes können keine Zeiger bzw. Referenzen deklariert werden.*

***Hinweis:*** *pointer wird nur in C/C++ verwendet und gibt an, wie viele Zeigerebenen die Variable hat. Handelt es sich beispielsweise um einen Zeiger auf ein Objekt, ist pointer 1. Ist es ein Zeiger auf einen Zeiger, ist pointer 2.*

### *Funktionen/Methoden/Operatoren/Casts*

```
<function>
    <type callby="reference|value" pointer="[int]">
        [...]
    </type>
    <code>
        [...]
    </code>
</function>
```

***Hinweis:*** *Die Eigenschaften callby und pointer werden nur in der Standardbibliothek verwendet. In normalen Codes können keine Zeiger bzw. Referenzen deklariert werden.*

***Hinweis:*** *pointer wird nur in C/C++ verwendet und gibt an, wie viele Zeigerebenen die Variable hat. Handelt es sich beispielsweise um einen Zeiger auf ein Objekt, ist pointer 1. Ist es ein Zeiger auf einen Zeiger, ist pointer 2.*

### *If*

```
<if-block>
    <if>
        <condition>
            [...]
        </condition>
        <code>
            [...]
        </code>
    </if>
    <else-if>
        <condition>
            [...]
        </condition>
        <code>
            [...]
        </code>
    </else-if>
    <else>
        <code>
            [...]
        </code>
    </else>
</if-block>
```

### *Select*

```
<select>
    <case>
        <value>[...]</value>
        <value>[...]</value>
        <code>[...]</code>
    </case>
```

```
        <case>
            <value>[...]</value>
            <code>[...]</code>
        </case>
        <default>
            <code>[...]</code>
        </default>
    </select>
```

***While***

```
    <while>
        <condition>
            [...]
        </condition>
        <code>
            [...]
        </code>
    </while>
```

***Repeat***

```
    <repeat>
        <code>
            [...]
        </code>
        <until>
            [...]
        </until>
    </repeat>
```

***For (mit Zähler)***

```
    <count>
        <from>
            [...]
        </from>
        <to>
            [...]
        </to>
```

```
        <code>
            [...]
        </code>
    </count>
```

***Loop (allgemeine Schleife)***

```
    <loop>
        <init>
            [...]
        </init>
        <end>
            [...]
        </end>
        <count>
            [...]
        </count>
        <code>
            [...]
        </code>
    </loop>
```

***Template***

```
    <template>
        <parameters>
            [...]
        </parameters>
        <code>
            [...]
        </code>
    </template>
```

***Zuweisungen***

```
    <assign>
        <var>
            [...]
        </var>
        <value>
```

```
            [...]
        </value>
    </assign>
```

***Vergleich***

```
    <compare>
        <term> [...] </term>
        <term> [...] </term>
    </compare>
```

***Addition***

```
    <add>
        <term> [...] </term>
        <term> [...] </term>
    </add>
```

***Subtraktion***

```
    <subtract>
        <term> [...] </term>
        <term> [...] </term>
    </subtract>
```

***Division***

```
    <divide>
        <term> [...] </term>
        <term> [...] </term>
    </divide>
```

***Potenzieren***

```
    <pow>
        <term> [...] </term>
        <term> [...] </term>
    </pow>
```

***Logisches Und***

```
    <and>
        <term> [...] </term>
        <term> [...] </term>
```

```
</and>
```

### *Logisches Oder*

```
<or>
    <term> [...] </term>
    <term> [...] </term>
</or>
```

### *Binäres Und*

```
<bin-and>
    <term> [...] </term>
    <term> [...] </term>
</bin-and>
```

### *Binäres Oder*

```
<bin-or>
    <term> [...] </term>
    <term> [...] </term>
</bin-or>
```

### *Xor*

```
<xor>
    <term> [...] </term>
    <term> [...] </term>
<xor>
```

### *Strings*

```
<string>
    [string]
</string>
```

***Hinweis:*** *<string>Hello World</string> wird in Hello World mit Anführungszeichen umgewandelt.*

### *Funktions-/Methodenaufruf*

```
<call>
    <function> [...] </function>
    <from> [...] </from>
</call>
```

***Return***

```
<return>
    […]
</return>
```

***Klassendefinition***

```
<class name="[string]">
    <public>
        <definitions>
            […]
        </definitions>
        <vars>
            […]
        </vars>
        <methods>
            […]
        </methods>
        <operators>
            […]
        </operators>
        <casts>
            […]
        </casts>
    </public>
    <protected>
        <definitions>
            […]
        </definitions>
        <vars>
            […]
        </vars>
        <methods>
            […]
        </methods>
        <operators>
            […]
        </operators>
```

```
            <casts>
                [...]
            </casts>
        </protected>
        <private>
            <definitions>
                [...]
            </definitions>
            <vars>
                [...]
            </vars>
            <methods>
                [...]
            </methods>
            <operators>
                [...]
            </operators>
            <casts>
                [...]
            </casts>
        </private>
    </class>
```

***Typedef***

```
    <define>
        <term> [type] </term>
        <alias> [...] </alias>
    </define>
```

***#define***

```
    <define>
        <term> [...] </term>
        <alias> [...] </alias>
    </define>
```

***#ifdef***

```
    <ifdef-block>
```

```
    <ifdef>
        <condition> [...] </condition>
        <code> [...] </code>
    </ifdef>
    <else-ifdef>
        <condition> [...] </condition>
        <code> [...] </code>
    </else-ifdef>
    <else>
        <code> [...] </code>
    </else>
</ifdef-block>
```

### *Import*

```
<import>
    [module]
</import>
```

### *Break*

```
<break>
    [loop]
</break>
```

***Hinweis:*** *loop ist eine optionale Eigenschaft. Sie kann verwendet werden, um in die nächsthöhere Schleife zu springen.*

### *Continue*

```
<continue>
    [loop]
</continue>
```

***Hinweis:*** *loop ist eine optionale Eigenschaft. Sie kann verwendet werden, um in die nächsthöhere Schleife zu springen.*

### *Lokaler Variablenblock*

```
<local>
    [...]
</local>
```

### *Globaler Variablenblock*

```
<global>
```

```
    [...]
</global>
```

### *Klasseninterner Variablenblock*

```
<local>
    [...]
</local>
```

### *Konstruktor*

```
<constructor>
    [...]
</constructor>
```

***Hinweis:*** *Siehe Funktionsdeklaration.*

### *Destruktor*

```
<destructor>
    [...]
</destructor>
```

***Hinweis:*** *Siehe Funktionsdeklaration.*

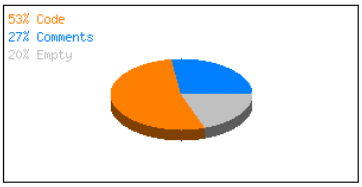
# Referenzen

## *Blitzprog Code Stats*

Code statistics generated by Blitzprog Code Stats 0.1.0 (www.blitzprog.de)

Statistics | Analysis | ToDo | Types | Functions | Includes

### Statistics




| File | Lines | Comment lines | Empty lines | Chars | Chars per line | Types | Functions | Includes | ToDo |
|---|---|---|---|---|---|---|---|---|---|
| INI.bmx | 213 | 58 | 52 | 4663 | 22 | 2 | 1 | 2 | 0 |
| XML.bmx | 401 | 154 | 78 | 9887 | 25 | 2 | 2 | 1 | 4 |
| RSS.bmx | 260 | 93 | 72 | 5283 | 20 | 2 | 3 | 1 | 1 |
| XUL.bmx | 510 | 132 | 98 | 12758 | 25 | 4 | 5 | 5 | 6 |
| Log.bmx | 93 | 38 | 25 | 2397 | 26 | 1 | 1 | 2 | 0 |
| Undo.bmx | 101 | 34 | 24 | 2764 | 27 | 1 | 1 | 2 | 0 |
| Macro.bmx | 106 | 35 | 25 | 2549 | 24 | 1 | 5 | 1 | 1 |
| Settings.bmx | 101 | 38 | 25 | 2645 | 26 | 0 | 0 | 1 | 0 |
| GadgetEx.bmx | 177 | 50 | 31 | 5358 | 30 | 0 | 13 | 3 | 0 |
| ProjectManagement.bmx | 342 | 66 | 79 | 7678 | 22 | 4 | 3 | 6 | 1 |
| ProgLang.bmx | 712 | 174 | 140 | 18779 | 26 | 7 | 9 | 7 | 0 |
| StringEx.bmx | 82 | 36 | 24 | 2107 | 26 | 0 | 2 | 1 | 1 |

## *Blitzprog Profiler*

Profile info generated by Blitzprog Profiler 0.2.5 (www.blitzprog.com)

| | | | | | |
|---|---|---|---|---|---|
| .Interpolate: | 0 ms | Total: | 0 ms | Calls: | 81 |
| .PositionEntityInterpolate: | 0 ms | Total: | 1 ms | Calls: | 27 |
| .SetBoundingBoxSize: | 1914 ms | Total: | 1914 ms | Calls: | 1 |
| .Show3DPoint: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .Show3DLine: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .DrawWireframe: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .CheckCull2D: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .UpdateNormalsFlat: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .PanoramaScreenshot: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .LoadMap: | 470 ms | Total: | 470 ms | Calls: | 1 |
| .LoadMapInfo: | 12 ms | Total: | 12 ms | Calls: | 1 |
| .SaveMapInfo: | 3 ms | Total: | 3 ms | Calls: | 1 |
| .CreateMapObject: | 958 ms | Total: | 1916 ms | Calls: | 2 |
| .DeleteMapObject: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .LoadSettings: | 1 ms | Total: | 1 ms | Calls: | 1 |
| .FPS: | 0 ms | Total: | 3 ms | Calls: | 910 |
| .LoadSkyBox: | 398 ms | Total: | 398 ms | Calls: | 1 |
| .MovePlayer: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .CreatePlayer: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .InitWindow: | 1140 ms | Total: | 1140 ms | Calls: | 1 |
| .UpdateMainMenu: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .Update3D: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .Update2D: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .UpdateInput: | 0 ms | Total: | 177 ms | Calls: | 910 |
| .CreateGame: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .DeleteGame: | 0 ms | Total: | 0 ms | Calls: | 0 |
| .CreateLog: | 2 ms | Total: | 2 ms | Calls: | 1 |
| .WriteToLog: | 22 ms | Total: | 534 ms | Calls: | 24 |
| .GetBoolString: | 0 ms | Total: | 0 ms | Calls: | 3 |
| .UpdateGUI: | 17 ms | Total: | 15625 ms | Calls: | 910 |
| .LoadGUIStyle: | 65 ms | Total: | 65 ms | Calls: | 1 |
| .CreateSlider: | 0 ms | Total: | 0 ms | Calls: | 9 |
| .CreateTextField: | 1 ms | Total: | 1 ms | Calls: | 1 |
| .CreateCheckBox: | 0 ms | Total: | 1 ms | Calls: | 5 |
| .CreatePanel: | 0 ms | Total: | 0 ms | Calls: | 1 |
| .CreateWindow: | 0 ms | Total: | 0 ms | Calls: | 2 |
| .UpdatePanels: | 0 ms | Total: | 3 ms | Calls: | 910 |
| .UpdateWindows: | 10 ms | Total: | 9210 ms | Calls: | 910 |

# *Physica Phenomena*