# IpsSkript: Aufbau der Skriptsprache

## Aufbau der Skriptsprache

Als Basis der Skriptsprache dient FastSkript. Als Sprachdialekt dient ‚Pascal-Skript‘, weil hiermit die internen Objekte unmittelbar abgebildet und bedient werden können.

Um den Skriptverlauf variabel gestalten zu können, kommen Variablen zum Einsatz, welche vom Typ

* Integer
* Boolean
* String
* TDateTime oder
* Objekte (siehe unten)

sein können. Hiervon wird rege Gebrauch gemacht, damit Angaben nur an einer Stelle definiert zu werden brauchen.

## Objekt IpsMaster

### Constructor Create: Objekt erzeugen

Bevor mit dem Skriptkern gearbeitet werden kann, muss das Objekt TIpsMaster erzeugt und die Referenz in einer Variablen gespeichert werden, damit hierauf im weiteren Verlauf zugegriffen werden kann:

**Beispiel:**

Program TestProgram;

Var

oMaster:TIpsMaster;

Begin

oMaster:=TIpsMaster.Create;

. . .

oMaster.Free;

end;

### Destructor Free: Objekt freigeben

Nachdem alle Skriptaktivitäten getätigt wurden, muss das Objekt wieder freigegeben werden. Dies bewirkt, dass sowohl der Speicher als auch die Datenbankverbindungen nicht mehr belegt sind. Dies geschieht durch die Anweisung: oMaster.Free (siehe obiges Beispiel).

### Funktion GetIni(<parameterbezeichnung>):String;

Mit Hilfe der Methode GetIni ist es möglich, auf eine zentrale Parameterdatei im INI-Format zuzugreifen. Dies ist sinnvoll, damit verschiedene Skripts auf zentrale Parameter gezielt und einheitlich zugreifen können, ohne dass diese an verschiedenen Stellen definiert zu werden brauchen und auch bei späteren Änderungen in allen Skripten immer aktuell bleiben.

Die INI-Datei befindet sich im gleichen Verzeichnis wie das Programm IpsSkript und hat den Namen ‚IpsSkript.ini‘. Sie hat folgenden Aufbau:

[Datei]

Variante=XYZ

[XYZ]

<parameterbezeichnung1>=<parameterstringwert1>

<parameterbezeichnung2>=<parameterstringwert2>

. . .

<parameterbezeichnungn>=<parameterstringwertn>

Durch diesen Aufbau ist es möglich, verschiedene Varianten von Parameterwerten vorzuhalten und durch Eintrag unter Datei;Variante… anzusprechen.

Im Skriptprogramm können die Parameter wie folgt angesprochen werden:

sVariante:=oMaster.GetIni('Datei;Variante');

sParameterStringWert1:=oMaster.GetIni(sVariante+';<parameterbezeichnung1>);

sParameterStringWert2:=oMaster.GetIni(sVariante+';<parameterbezeichnung2>);

Die Parameterbezeichnung ist zweigeteilt durch einen Semikolon. Der erste Parameter bezeichnet den Abschnitt, der in der INI-Datei durch eine eckige Klammer gekennzeichnet ist. Der zweite Parameter verweist auf die Variable innerhalb des Abschnittes, welche vor dem Gleichheitszeichen steht. Die Funktion GetINI(<parameterbezeichnung>) liefert den <parameterstringwert> zu der gewünschten <parameterbezeichnung> im Stringformat zurück.

### Methode SetPar(<parameterbezeichnung>,<parameterstring>);

Mit Hilfe der Methode SetPar ist es möglich, dem Skriptkern Parameter mitzuteilen. Im folgenden werden die Parameter und deren jeweilige Bedeutung erklärt. Der erste Parameter der Methode bestimmt die Informationsart, der zweite Parameter deren Wert.

|  |  |  |
| --- | --- | --- |
| Code | Parameter | Bedeutung |
| LogDatei | Pfad der Logdatei | Datei, in die Logausgabe erfolgen soll, wenn dies mit Code MitLogDatei gewünscht ist. |
| MitLogDatei | 0/1 | Merker, ob Logging in Datei aktiv ist (1) oder nicht (0). |
| MitLogDebug | 0/1 | Merker, ob Logging in Debug-Viewer aktiv ist (1) oder nicht (0). Sytemmeldungen werden IMMER im Debug-Viewer angezeigt. |
| Variante | EV / HWB / LINEG / SEF | Hiermit wird die Art der internen Verarbeitung beim Import gesteuert. |
| DataDir | <Pfad-Angabe> | Basis-Verzeichnis für Import. |
| DbParam | DbType,<typ> | Datenbank-Typ: ADOMDB, MSSQL, ORACLE, MYSQL, SQLITE, POSTGRE |
|  | DbServer,<name> | Name oder IP-Adresse des Datenbank-Servers |
|  | DbDatabase,<name> | Name der Datenbank |
|  | DbDatabaseFile,<dateiname> | Dateiname bei dateibasierten Datenbanken |
|  | DbUser,<name> | Benutzername für Zugang zum Datenbanksystem |
|  | DbPassword,<kennwort>  (kann nicht ausgelesen werden) | Kennwort für den Zugang zum Datenbanksystem |
|  | DbDirekt,<0/1> | Direkter Zugriff (Oracle) ohne Clientsoftware (über TCP/IP) |
|  | DbPort,<portnummer> | Portnummer bei direkten Zugriff |
|  | DbSID,<SID> | System-ID (Oracle) |
| Skriptdatei  (nur lesen) | „“ | Name der Skriptdatei |
| Error  (nur lesen) | „“ | Letzte Fehlermeldung |
| DateTime  (nur lesen) | „“ | Aktuelles Datum/Uhrzeit im Normformat yyyymmddhhnnss |
| DateTime2  (nur lesen) | „“ | Aktuelles Datum/Uhrzeit im Klartext-Format dd.mm.yyyy hh:nn |
| JobTime  (nur lesen) | „“ | Verbrauchte Zeit seit Beginn der Skriptverarbeitung. |
|  |  |  |

### Methode Log(<text>);

Der angegebene Text wird in die aktiven Log-Medien geschrieben. Welches Medium aktiv ist, hängt von der Parametrierung mit Hilfe der Methode SetPar (s.o.) ab.

### Methode LogDb/LogMsg/LogDebug(<text>);

Im Gegensatz zur Log-Methode wird der Text immer in das entsprechend Medium ausgegeben. Medien sind: Db=Logging-Tabelle der Datenbank, Msg=Dialogbox, Debug=Debug-Viewer.

### Methode EXECUTE (<Kommando>,<Parameter>);

|  |  |  |
| --- | --- | --- |
| Kommando | Parameter | Bedeutung |
| MDBCOPY | Muster-Datenbank | Die aktuelle Datenbankdatei (ADOMDB) wird aus einer Kopie neu erzeugt. Wird vor allem für Testzwecke benötigt. |
| SQLEXECUTE | SQL-Statement | Die angegebene SQL-Anweisung wird in der aktuellen Datenbank ausgeführt. |
| ConvertIpsDb | „“ | Die Datenbank wird auf die in der Software aktulle Datenbankversion upgegradet. |
| COPYTABLE |  |  |
| CLEAR |  |  |
| IPS  (siehe eigene Liste) | In Abhängigkeit von der ausgeführten Funktion | Es wird eine vereinbarte, vorhandene Funktion in der Geschäftslogik angestoßen. |
| ADHOC  (siehe eigene Liste) | In Abhängigkeit von der ausgeführten Funktion | Es wird eine vereinbarte, vorhandene Funktion in der Geschäftslogik angestoßen. |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| IPS-Funktion |  |  |
|  |  |  |
| Kommando | Parameter | Bedeutung |
| MDBCOPY | Muster-Datenbank | Die aktuelle Datenbankdatei (ADOMDB) wird aus einer Kopie neu erzeugt. Wird vor allem für Testzwecke benötigt. |
| SQLEXECUTE | SQL-Statement | Die angegebene SQL-Anweisung wird in der aktuellen Datenbank ausgeführt. |
| MDBCOPY | Muster-Datenbank | Die aktuelle Datenbankdatei (ADOMDB) wird aus einer Kopie neu erzeugt. Wird vor allem für Testzwecke benötigt. |
| ConvertIpsDb | „“ | Die Datenbank wird auf die in der Software aktulle Datenbankversion upgegradet. |
| IPS  (siehe eigene Liste) | In Abhängigkeit von der ausgeführten Funktion | Es wird eine vereinbarte, vorhandene Funktion in der Geschäftslogik angestoßen. |
| ADHOC  (siehe eigene Liste) | In Abhängigkeit von der ausgeführten Funktion | Es wird eine vereinbarte, vorhandene Funktion in der Geschäftslogik angestoßen. |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| ADHOC-Funktion | Parameter | Bedeutung |
| LISTOLIX | Exportdatei | Die Parameter aus dem ehemals aktuellen LIS-Format werden in das Standard-LIX-Format (\*.lix.csv) exportiert. |
| LIXEXPORT | Exportdatei | Die interne LIX-Liste wird als Standard-LIX-Format ausgegeben (\*.lix.csv). |
| LIXIMPORT  LIXIMPORTCLEAR | Importdatei | Die LIX-Liste der aktuellen Datenbank wird mit den Listenwerten aus der Austauschdatei im Standard-LIX-Format gefüllt. Bei „LIXIMPORTCLEAR“ wird zudem vorher die bestehende Liste geleert. |
|  |  |  |
|  |  |  |

## **Objekt** TData