FMXML Developer Reference: Scripts, Tables, and Fields

Version 1.4 — May 2026

Ivan M. Granger Codence / Fullcity

Table of Contents

FMXML Developer Reference: Scripts, Tables, and Fields 1
- Introduction 3
  - What Is FMXML? 3
  - What FMXML Handles — and What It Doesn't 3
  - How to Get FMXML from FileMaker 3
  - Clipboard Access Tools 4
  - How to Use This Document 4
- Part 1 — The fmxmlsnippet Wrapper 5
  - Two Modes: Full Objects vs. Partial Content 5
- Part 2 — Tables and Fields 6
  - Full Table Structure 6
  - Fields-Only Structure 7
  - Anatomy of a Field Element 7
  - Standard Fields 7
  - Auto-Enter Options 8
  - Auto-Enter by Calculation 9
  - Calculation Fields 10
  - Summary Fields 10
- Part 3 — Scripts 11
  - Full Script Structure 11
  - Steps-Only Structure 11
  - Anatomy of a Step Element 11
  - Working with IDs 12
  - Structural Patterns 12
- What This Document Doesn't Cover (Yet) 14

Appendix A — Script Step Reference 15
Control Flow 15
Variables 16
Fields 17
Records 18
Navigation 19
Scripts 21
Utility 22
Appendix B — Comprehensive Step ID Catalog 25

Introduction

What Is FMXML?

FileMaker stores scripts, tables, fields, layouts, and other schema objects internally as XML. Most developers never encounter this XML directly — FileMaker handles it invisibly in the background. But it's always there, and it's accessible.

When you copy a script, a set of fields, or other schema objects from FileMaker to the clipboard, what actually gets placed on the clipboard is a structured XML representation of those objects — commonly called FMXML. This works in reverse, as well: FMXML can be pasted back into FileMaker to recreate those objects. This is what makes it possible to copy and paste most schema elements in FileMaker. Copy a script from one file, paste it into another.

With access to the right tools, you can intercept the FMXML hidden on the clipboard, modify it, and put it back on the clipboard and paste the modified code back into FileMaker as new schema.

Understanding the format opens up new possibilities for how you build and manage FileMaker solutions.

Because Claris has not released official documentation on FMXML, this guide outlines the fundamental structure and patterns of FMXML so you can read it – and modify it – with confidence.

What FMXML Handles — and What It Doesn't

It's worth establishing up front what the FileMaker clipboard can and cannot do with FMXML.

What the clipboard handles: - Scripts and script steps - Tables and fields - Custom functions - Layout objects (fields, buttons, portals, labels, and other layout elements) - Value lists - Themes

What the clipboard does not handle: - The relationship graph — table occurrences (TOs) and relationships, since they cannot be copied and pasted within FileMaker.

How to Get FMXML from FileMaker

FileMaker writes FMXML to the clipboard whenever you copy script steps, scripts, fields, or other schema objects.

To get the FMXML for any object:

1. Select and copy the object(s) in FileMaker — script steps in the Script Editor, fields in the Manage Database dialog, etc.
2. The clipboard now contains FMXML.

3. Use a utility like FM Clipboard Tool to read, display, and edit the FMXML from the clipboard.

The XML on the clipboard is not directly visible in the normal way — it's stored in a special clipboard format that FileMaker recognizes, not as plain text. A utility like FM Clipboard Tool can decode that format, display the FMXML as readable text, and let you edit it before pasting back. This visibility is what makes it practical to inspect, learn from, and hand-author FMXML.

Clipboard Access Tools

Because FMXML is stored on the clipboard in a proprietary binary format — not as plain text — you need a tool that understands FileMaker's clipboard structure in order to read and write it.

Several utilities are available:

FM Clipboard Tool (Codence / Fullcity) — A FileMaker utility file that reads clipboard FMXML into an editable text field and writes it back in the correct binary format. Uses the free BaseElements plugin, which is widely used in FileMaker development environments and provides the clipboard read/write functions the tool depends on. Runs on Mac and Windows; requires FileMaker Pro 18 or later.

2empowerFM Clipboard Explorer (2empowerFM) — An alternative clipboard utility that provides similar access to clipboard FMXML. Uses its own 2empowerFM plugin.

Plugin-free alternatives exist, as well. Watch for information from Codence and Fullcity on AI development methodologies for working with FileMaker.

How to Use This Document

This document is organized in three parts followed by two appendices:

- **Part 1** covers the outer wrapper structure shared by all FMXML snippets
- **Part 2** covers tables and fields — structure, field types, and common options
- **Part 3** covers scripts — structure, how IDs work, and the structural patterns you'll see throughout
- **Appendix A** is the script step reference — common steps with plain text and XML representations
- **Appendix B** is a comprehensive catalog of all FileMaker script steps with their IDs

Parts 1–3 are for reading. The appendices are for looking things up.

This document covers tables, fields, scripts, and script steps — the element types most developers encounter first. Custom functions, layout objects, value lists, and themes are not covered in this version; see *What This Document Doesn't Cover* at the end.

Part 1 — The fmxmlsnippet Wrapper

All FileMaker clipboard XML is wrapped in a single root element:

```
<fmxmlsnippet type="FMObjectList">
    <!-- content goes here -->
</fmxmlsnippet>
```

The `type="FMObjectList"` attribute is fixed — it's the same for every FMXML snippet regardless of what's inside. It tells FileMaker that the contents are a list of objects to be pasted.

Two Modes: Full Objects vs. Partial Content

FMXML operates in two distinct modes depending on what you're pasting.

Full objects — When pasting a complete script or a complete table with its fields, the outer object element (`<Script>` or `<BaseTable>`) is included inside the wrapper:

```
<fmxmlsnippet type="FMObjectList">
    <Script id="0" name="My Script">
        <!-- steps go here -->
    </Script>
</fmxmlsnippet>
```

Partial content — When pasting individual script steps into an existing script, or fields into an existing table, the wrapper contains the steps or fields directly — no `<Script>` or `<BaseTable>` wrapper:

```
<fmxmlsnippet type="FMObjectList">
    <!-- steps or fields go here directly -->
</fmxmlsnippet>
```

This distinction matters. If you include a `<Script>` wrapper when you intend to paste steps, FileMaker will create a new script rather than inserting steps into the current one. Likewise, including a `<BaseTable>` wrapper creates a new table rather than adding fields to an existing one.

Part 2 — Tables and Fields

Important — FileMaker is silently strict about field-paste payloads

FileMaker's paste handler accepts FMXML byte-by-byte against a fixed internal vocabulary. Two patterns silently degrade or corrupt the resulting fields without any error reported:

- **Field `id` must be a positive integer.** Using `id="0"` inside a `<BaseTable>` wrapper causes FileMaker to silently drop the field from the field catalog while still incrementing the catalog's count. The table appears in Manage Database with one or more fields missing — and any subsequent script step or layout object that references the dropped field by name will throw `[102] Field is missing` at runtime. Always use sequential positive integers (`"1"`, `"2"`, …) within each `<BaseTable>`.
- **Attribute literals are case-sensitive.** Values like `dataType` and AutoEnter `value` must match FileMaker's canonical form exactly. For example, `dataType="TimeStamp"` is correct (capital S in the middle); `dataType="Timestamp"` is silently rejected and the field is created as a default `Number` field with no auto-enter. The same rule applies to `CreationTimeStamp` and `ModificationTimeStamp`.

The pattern in both cases is the same: FileMaker accepts the paste, the operation looks successful, and the corruption only surfaces later when something fails to find a field that "should be there." The examples throughout this section all follow the canonical conventions.

Full Table Structure

To create a new table with fields, use the `<BaseTable>` wrapper:

```
<fmxmlsnippet type="FMObjectList">
    <BaseTable name="Contacts">
        <!-- fields go here -->
    </BaseTable>
</fmxmlsnippet>
```

Multiple tables can be included in a single snippet:

```
<fmxmlsnippet type="FMObjectList">
    <BaseTable name="Contacts">
        <!-- Contacts fields -->
    </BaseTable>
    <BaseTable name="Companies">
        <!-- Companies fields -->
    </BaseTable>
</fmxmlsnippet>
```

Fields-Only Structure

To add fields to an existing table, omit the `<BaseTable>` wrapper. Fields go directly inside the `fmxmlsnippet`:

```
<fmxmlsnippet type="FMObjectList">
    <!-- fields go here directly -->
</fmxmlsnippet>
```

FileMaker pastes the fields into whichever table is currently active in the Manage Database dialog.

Anatomy of a Field Element

Every field begins with a `<Field>` opening tag:

```
<Field id="1" dataType="Text" fieldType="Normal" name="Name First">
```

Attribute	Description
`id`	A positive integer. Use sequential ids (`"1"`, `"2"`, ...) within each `<BaseTable>`. Never `"0"` — see the callout at the top of Part 2
`dataType`	The field's data type: `Text`, `Number`, `Date`, `Time`, `TimeStamp`, `Container` (note `TimeStamp` casing)
`fieldType`	The field's behavior type: `Normal`, `Calculated`, `Summary`
`name`	The field name as it will appear in the table

A complete field element also includes child elements for comments, auto-enter options, validation rules, and storage settings.

Standard Fields

Standard fields (Text, Number, Date, Time, TimeStamp, and Container) share this structure:

```
<Field id="1" dataType="Text" fieldType="Normal" name="Name First">
    <Comment>First name</Comment>
    <AutoEnter allowEditing="True" constant="False" furigana="False" lookup="
False" calculation="False">
        <ConstantData></ConstantData>
    </AutoEnter>
    <Validation type="OnlyDuringDataEntry">
        <NotEmpty value="False"></NotEmpty>
        <Unique value="False"></Unique>
        <Existing value="False"></Existing>
        <StrictValidation value="False"></StrictValidation>
    </Validation>
    <Storage autoIndex="True" index="None" global="False" maxRepetitions="1">
```

```
</Storage>
</Field>
```

<Comment> — The field comment. Optional; omit if not needed.

<AutoEnter> — Controls auto-enter behavior. When no auto-enter option is set, all attributes are `"False"` and `<ConstantData>` is empty, as shown above. See the sections below for auto-enter variants.

<Validation> — Validation rules. The `type` attribute is `"OnlyDuringDataEntry"` or `"Always"`. The child elements correspond to the checkboxes in FileMaker's field validation dialog — set each to `"True"` or `"False"` as needed.

<Storage> — Indexing and storage settings: - `autoIndex="True"` — FileMaker manages the index automatically (creates indexes on demand as needed) - `index="None"` — index mode. Valid values are `"None"`, `"All"`, `"Minimal"`, and `"Value"`. The FileMaker UI default is `"None"` (paired with `autoIndex="True"`) — indexes are created on demand rather than always maintained. Emitting `"All"` instead forces always-indexed, which differs from UI default and can slow large imports and increase file size unnecessarily - `global="True"` — marks the field as a global (single value stored across all records) - `maxRepetitions="1"` — `"1"` for non-repeating fields; higher values for repeating fields (note: plural — `maxRepetitions`, not `maxRepetition`) - `indexLanguage="English"` — optional; omit when not customizing the index language. FileMaker fills in a `<LanguageReference>` child element automatically on paste when the attribute is absent. Include the attribute only when explicitly selecting a non-default language

Container fields use `dataType="Binary"`.

Auto-Enter Options

Auto-enter a constant value:

```
<AutoEnter allowEditing="True" constant="True" furigana="False" lookup="False
" calculation="False">
    <ConstantData>Default Value</ConstantData>
</AutoEnter>
```

Set `allowEditing="False"` if the user should not be able to override the auto-entered value.

Auto-enter creation or modification metadata:

```
<AutoEnter allowEditing="False" value="CreationTimeStamp" constant="False" fu
rigana="False" lookup="False" calculation="False">
    <ConstantData></ConstantData>
</AutoEnter>
```

Common `value` options:

Value	Description
`CreationDate`	Date the record was created
`CreationTime`	Time the record was created
`CreationTimeStamp`	Date and time the record was created (note capital S)
`CreatorName`	Full name of the account that created the record
`CreatorAccountName`	Account name that created the record
`ModificationDate`	Date the record was last modified
`ModificationTimeStamp`	Date and time of last modification (note capital S)
`ModifierAccountName`	Account name that last modified the record

Set `allowEditing="False"` when the value should be set once and not changed.

Auto-Enter by Calculation

A standard (Normal) field with an auto-enter calculation uses `fieldType="Normal"` and sets `calculation="True"` on the `<AutoEnter>` element:

```
<Field id="1" dataType="Text" fieldType="Normal" name="Status">
    <AutoEnter allowEditing="True" overwriteExistingValue="False" alwaysEvalu
ate="False"
               constant="False" furigana="False" lookup="False" calculation="
True">
        <ConstantData></ConstantData>
        <Calculation table="Contacts"><![CDATA["Active"]]></Calculation>
    </AutoEnter>
    <Validation type="OnlyDuringDataEntry">
        <NotEmpty value="False"></NotEmpty>
        <Unique value="False"></Unique>
        <Existing value="False"></Existing>
        <StrictValidation value="False"></StrictValidation>
    </Validation>
    <Storage autoIndex="True" index="None" global="False" maxRepetitions="1">
</Storage>
</Field>
```

Key attributes on `<AutoEnter>` for calculation fields:

- `calculation="True"` — enables the auto-enter calculation
- `overwriteExistingValue="False"` — preserves existing values; equivalent to checking “Do not replace existing value” in FileMaker’s field options. Set to `"True"` to always recalculate on entry
- `alwaysEvaluate="False"` — when `"True"`, the calculation re-evaluates whenever any referenced field changes (similar to an unstored calculation). Usually `"False"`
- The `<Calculation>` child contains the formula; the `table` attribute sets the evaluation context

Calculation Fields

Calculation fields use `fieldType="Calculated"` and include the formula directly as a child element:

```
<Field id="1" dataType="Text" fieldType="Calculated" name="Name Full">
    <Calculation table="Contacts"><![CDATA[Name First & " " & Name Last]]></C
alculation>
    <Comment>Full name derived from first and last name fields</Comment>
    <AutoEnter alwaysEvaluate="False"></AutoEnter>
    <Storage storeCalculationResults="True" autoIndex="True" index="None" glo
bal="False" maxRepetitions="1"></Storage>
</Field>
```

- `dataType` is the result type — `Text`, `Number`, `Date`, `Time`, `TimeStamp`, or `Container`
- The `table` attribute on `<Calculation>` sets the evaluation context (the table name)
- `storeCalculationResults="True"` stores the result; `"False"` evaluates on demand (unstored)
- `alwaysEvaluate` on `<AutoEnter>` defaults to `"False"` for stored calculations

Summary Fields

Summary fields aggregate data across a found set. They use `fieldType="Summary"`:

```
<Field id="1" dataType="Number" fieldType="Summary" name="Total Amount">
    <SummaryInfo operation="Total" summarizeRepetition="1" restartForEachSort
edGroup="False">
        <SummaryField>
            <Field id="1" name="Amount"></Field>
        </SummaryField>
    </SummaryInfo>
    <Comment>Total of Amount across the found set</Comment>
</Field>
```

- `operation` values: `Total`, `Average`, `Count`, `Minimum`, `Maximum`, `StandardDeviation`, `Fraction of Total`
- `<SummaryField>` identifies the field being summarized — use `id="1"` as a placeholder; FileMaker resolves by name
- `restartForEachSortedGroup="True"` corresponds to the “Restart summary for each sorted group” option

---

Part 3 — Scripts

Full Script Structure

A complete script uses the `<Script>` element as a direct child of the wrapper:

```
<fmxmlsnippet type="FMObjectList">
    <Script id="0" name="Initialize Session">
        <!-- steps go here -->
    </Script>
</fmxmlsnippet>
```

- Set `id="0"` — FileMaker assigns the actual ID on paste
- The `name` attribute becomes the script's name in the Scripts list
- Multiple scripts can be included in a single snippet — each as a separate `<Script>` element

Steps-Only Structure

To paste steps into an existing script, omit the `<Script>` wrapper entirely:

```
<fmxmlsnippet type="FMObjectList">
    <Step enable="True" id="85" name="Allow User Abort">
        <Set state="False"></Set>
    </Step>
    <Step enable="True" id="86" name="Set Error Capture">
        <Set state="True"></Set>
    </Step>
</fmxmlsnippet>
```

Steps are pasted at the current cursor position in the open Script Editor.

Anatomy of a Step Element

Every script step is represented by a `<Step>` element with three core attributes:

```
<Step enable="True" id="68" name="If">
```

Attribute	Description
`enable`	`"True"` = step is active; `"False"` = step is disabled (grayed out in the Script Editor)
`id`	FileMaker's internal integer ID for this step type — must be exact
`name`	The human-readable step name — present for readability, but `id` is authoritative

Working with IDs

IDs appear in three places in script FMXML, and they work differently in each.

Step IDs are the `id` attribute on every `<Step>` element. These must be exact — FileMaker uses the step ID to determine what kind of step to create. The `name` attribute is redundant; if the `id` is wrong, the wrong step will be created regardless of what `name` says. Step IDs are fixed integers assigned by FileMaker and do not change between files or versions. A complete catalog of step names and IDs is in Appendix B.

Field IDs appear in `<Field>` references within steps:

```
<Field table="Contacts" id="1" name="Name First"></Field>
```

These are the internal IDs of specific fields in a specific file. When writing FMXML by hand — or moving FMXML between files — you typically won't know the correct field ID. You can safely use `id="1"` as a placeholder. FileMaker resolves field references by name on paste, so as long as the `table` and `name` values are correct, the paste will work and FileMaker will fill in the real ID automatically.

Script IDs work the same way as field IDs. When a step references another script:

```
<Script id="1" name="Initialize Session"></Script>
```

Use `id="1"` as a placeholder. FileMaker resolves the reference by name.

The practical takeaway: step IDs must always be exact; field and script IDs can be placeholders.

Structural Patterns

Script steps follow a small set of structural patterns. Recognizing these makes it much easier to read unfamiliar FMXML.

Steps with no children — Steps that have no configurable options appear as an empty element:

```
New Record/Request
```

```
<Step enable="True" id="7" name="New Record/Request">
</Step>
```

Steps with a Calculation child — Steps that take a calculation as input wrap the formula in a `<Calculation>` element using CDATA:

```
If [Contacts::Status = "Active"]
```

```
<Step enable="True" id="68" name="If">
    <Calculation><![CDATA[Contacts::Status = "Active"]]></Calculation>
</Step>
```

CDATA (`<![CDATA[...]]>`) tells the XML parser to treat the content as literal text rather than XML markup. All calculations in FMXML are wrapped this way.

Steps with state attributes — Steps with on/off toggles use a child element with a `state` attribute:

```
Allow User Abort [Off]
```

```
<Step enable="True" id="85" name="Allow User Abort">
    <Set state="False"></Set>
</Step>
```

`state="True"` and `state="False"` correspond to the two states of a checkbox or toggle in the step's configuration dialog.

Steps with field references — Steps that target a specific field include a `<Field>` child element:

```
Set Field [Contacts::Name First; "Hello"]
```

```
<Step enable="True" id="76" name="Set Field">
    <Calculation><![CDATA["Hello"]]></Calculation>
    <Field table="Contacts" id="1" name="Name First"></Field>
</Step>
```

Steps with script references — Steps that call another script include a `<Script>` child element:

```
Perform Script ["Initialize Session"]
```

```
<Step enable="True" id="1" name="Perform Script">
    <Script id="1" name="Initialize Session"></Script>
</Step>
```

The step reference catalog in Appendix A shows these patterns applied across all common step types.

What This Document Doesn't Cover (Yet)

This is version 1.4 of what may become a growing resource. It focuses on the element types most developers encounter first — scripts, script steps, tables, and fields.

FileMaker's FMXML format covers additional schema object types not documented here:

- **Custom Functions** — reusable calculation formulas defined at the file level
- **Layout Objects** — fields, buttons, portals, text labels, tab panels, and other objects placed on layouts
- **Value Lists** — static and dynamic lists used to control data entry
- **Themes** — CSS-based styling definitions that govern the appearance of layout objects

Note that **relationships** are not included because the FileMaker clipboard does not support copying and pasting of table occurrences or relationships.

The element types listed above follow the same core FMXML principles covered here — a root `fmxmlsnippet` wrapper, consistent use of `id` attributes, and CDATA-wrapped calculations — so the patterns you've learned in this document apply across the board. Detailed coverage of these element types may appear in future versions.

Appendix A — Script Step Reference

Each step is shown with its plain text representation (as it appears in FileMaker's Script Editor) followed by the corresponding FMXML.

Control Flow

If (Step ID: 68)

```
If [Contacts::Status = "Active"]
```

```
<Step enable="True" id="68" name="If">
    <Calculation><![CDATA[Contacts::Status = "Active"]]></Calculation>
</Step>
```

Else If (Step ID: 125)

```
Else If [Contacts::Status = "Pending"]
```

```
<Step enable="True" id="125" name="Else If">
    <Calculation><![CDATA[Contacts::Status = "Pending"]]></Calculation>
</Step>
```

Else (Step ID: 69)

```
Else
```

```
<Step enable="True" id="69" name="Else">
    <Restore state="False"></Restore>
</Step>
```

End If (Step ID: 70)

```
End If
```

```
<Step enable="True" id="70" name="End If">
</Step>
```

Loop (Step ID: 71)

```
Loop
```

```
<Step enable="True" id="71" name="Loop">
    <FlushType value="Always"></FlushType>
</Step>
```

`<FlushType value="Always">` is the standard setting and should always be included.

Exit Loop If (Step ID: 72)

```
Exit Loop If [$counter > 100]
```

```
<Step enable="True" id="72" name="Exit Loop If">
    <Calculation><![CDATA[$counter > 100]]></Calculation>
</Step>
```

End Loop (Step ID: 73)

```
End Loop
```

```
<Step enable="True" id="73" name="End Loop">
</Step>
```

Exit Script (Step ID: 103)

With result:

```
Exit Script [Result: $result]
```

```
<Step enable="True" id="103" name="Exit Script">
    <Calculation><![CDATA[$result]]></Calculation>
</Step>
```

Without result:

```
Exit Script [Result: ""]
```

```
<Step enable="True" id="103" name="Exit Script">
    <Calculation><![CDATA[""]]></Calculation>
</Step>
```

Variables

Set Variable (Step ID: 141)

Local variable:

```
Set Variable [$fullName; Value: Contacts::Name First & " " & Contacts::Name L
ast]
```

```
<Step enable="True" id="141" name="Set Variable">
    <Value>
        <Calculation><![CDATA[Contacts::Name First & " " & Contacts::Name Las
t]]></Calculation>
    </Value>
    <Repetition>
        <Calculation><![CDATA[1]]></Calculation>
    </Repetition>
    <Name>$fullName</Name>
</Step>
```

Global variable:

```
Set Variable [$$currentUser; Value: Get(AccountName)]
```

```
<Step enable="True" id="141" name="Set Variable">
    <Value>
        <Calculation><![CDATA[Get(AccountName)]]></Calculation>
    </Value>
    <Repetition>
        <Calculation><![CDATA[1]]></Calculation>
    </Repetition>
    <Name>$$currentUser</Name>
</Step>
```

- `<Name>` contains the variable name including the $ (local) or $$ (global) prefix
- `<Repetition>` is 1 for non-repeating variables
- `<Value>` contains the calculation that produces the variable's value

---

Fields

Set Field (Step ID: 76)

```
Set Field [Contacts::Name Display; $fullName]
```

```
<Step enable="True" id="76" name="Set Field">
    <Calculation><![CDATA[$fullName]]></Calculation>
    <Field table="Contacts" id="1" name="Name Display"></Field>
</Step>
```

The `<Calculation>` is the value being set. The `<Field>` element identifies the target. The field name is the bare field name only — the table is specified separately in the `table` attribute.

Set Field By Name (Step ID: 147)

```
Set Field By Name ["Contacts::Name Display"; $value]
```

```
<Step enable="True" id="147" name="Set Field By Name">
    <Result><Calculation><![CDATA[$value]]></Calculation></Result>
    <TargetName><Calculation><![CDATA["Contacts::Name Display"]]></Calculatio
n></TargetName>
</Step>
```

Useful when the target field needs to be determined dynamically at runtime. The `<TargetName>` calculation should evaluate to a fully qualified `"TableName::FieldName"` string.

Records

New Record/Request (Step ID: 7)

```
New Record/Request
```

```
<Step enable="True" id="7" name="New Record/Request">
</Step>
```

Delete Record/Request (Step ID: 9)

Without dialog:

```
Delete Record/Request [No dialog]
```

```
<Step enable="True" id="9" name="Delete Record/Request">
    <NoInteract state="True"></NoInteract>
</Step>
```

With dialog:

```
Delete Record/Request
```

```
<Step enable="True" id="9" name="Delete Record/Request">
    <NoInteract state="False"></NoInteract>
</Step>
```

`NoInteract state="True"` suppresses the confirmation dialog.

Commit Records/Requests (Step ID: 75)

```
Commit Records/Requests [No dialog; Skip data entry validation: No]
```

```
<Step enable="True" id="75" name="Commit Records/Requests">
    <NoInteract state="True"></NoInteract>
    <Option state="False"></Option>
    <ESSForceCommit state="False"></ESSForceCommit>
</Step>
```

- `NoInteract`: `"True"` = no dialog (With dialog: Off)
- `Option`: `"True"` = skip data entry validation
- `ESSForceCommit`: `"True"` = force commit on external SQL source tables

Navigation

Go to Layout (Step ID: 6)

By layout name calculation (most common — supports dynamic layout names):

```
Go to Layout [Layout Name By Calc: "Contact Detail"]
```

```
<Step enable="True" id="6" name="Go to Layout">
    <LayoutDestination value="LayoutNameByCalc"></LayoutDestination>
    <Layout><Calculation><![CDATA["Contact Detail"]]></Calculation></Layout>
</Step>
```

By layout selected from list (static reference):

```
Go to Layout ["Contact Detail" (Contacts)]
```

```
<Step enable="True" id="6" name="Go to Layout">
    <LayoutDestination value="SelectedLayout"></LayoutDestination>
    <Layout id="0" name="Contact Detail"></Layout>
</Step>
```

Return to original layout:

```
Go to Layout [original layout]
```

```
<Step enable="True" id="6" name="Go to Layout">
    <LayoutDestination value="OriginalLayout"></LayoutDestination>
</Step>
```

The `LayoutDestination` value determines which variant is used. `LayoutNameByCalc` is the most flexible — the `<Layout>` child contains a calculation that can be a literal string or a dynamic expression.

Go to Record/Request/Page (Step ID: 16)

First record:

```
Go to Record/Request/Page [First]
```

```
<Step enable="True" id="16" name="Go to Record/Request/Page">
    <NoInteract state="True"></NoInteract>
    <RowPageLocation value="First"></RowPageLocation>
</Step>
```

RowPageLocation values: "First", "Last", "Previous", "Next", "ByCalculation"

Next record, exit after last:

```
Go to Record/Request/Page [Next; Exit after last]
```

```
<Step enable="True" id="16" name="Go to Record/Request/Page">
    <NoInteract state="True"></NoInteract>
    <Exit state="True"></Exit>
    <RowPageLocation value="Next"></RowPageLocation>
</Step>
```

The <Exit> element only applies when navigating "Next" or "Previous".

By calculation:

```
Go to Record/Request/Page [By Calculation: $recordNumber]
```

```
<Step enable="True" id="16" name="Go to Record/Request/Page">
    <NoInteract state="True"></NoInteract>
    <RowPageLocation value="ByCalculation"></RowPageLocation>
    <Calculation><![CDATA[$recordNumber]]></Calculation>
</Step>
```

Go to Field (Step ID: 17)

```
Go to Field [Contacts::Name First]
```

```
<Step enable="True" id="17" name="Go to Field">
    <SelectAll state="False"></SelectAll>
    <Field table="Contacts" id="1" name="Name First"></Field>
</Step>
```

With select all:

```
Go to Field [Select/perform; Contacts::Name First]
```

```
<Step enable="True" id="17" name="Go to Field">
    <SelectAll state="True"></SelectAll>
    <Field table="Contacts" id="1" name="Name First"></Field>
</Step>
```

`SelectAll state="True"` selects the entire contents of the field on entry.

Scripts

Perform Script (Step ID: 1)

Without parameter:

```
Perform Script ["Initialize Session"]
```

```
<Step enable="True" id="1" name="Perform Script">
    <Script id="1" name="Initialize Session"></Script>
</Step>
```

With parameter:

```
Perform Script ["Initialize Session"; Parameter: $parameter]
```

```
<Step enable="True" id="1" name="Perform Script">
    <Calculation><![CDATA[$parameter]]></Calculation>
    <Script id="1" name="Initialize Session"></Script>
</Step>
```

The `<Calculation>` element (the script parameter) comes before the `<Script>` reference. Omit it entirely when no parameter is needed.

Perform Script on Server (Step ID: 164)

Wait for completion:

```
Perform Script on Server [Wait for completion: On; "Process on Server"; Param
eter: $parameter]
```

```
<Step enable="True" id="164" name="Perform Script on Server">
    <WaitForCompletion state="True"></WaitForCompletion>
    <Calculation><![CDATA[$parameter]]></Calculation>
    <Script id="1" name="Process on Server"></Script>
</Step>
```

Fire and forget:

```
Perform Script on Server [Wait for completion: Off; "Process on Server"]
```

```
<Step enable="True" id="164" name="Perform Script on Server">
    <WaitForCompletion state="False"></WaitForCompletion>
    <Script id="1" name="Process on Server"></Script>
</Step>
```

`WaitForCompletion`: `"True"` = pause execution and wait for the server script to finish; `"False"` = fire and forget.

Utility

Allow User Abort (Step ID: 85)

Off (prevent abort):

```
Allow User Abort [Off]
```

```
<Step enable="True" id="85" name="Allow User Abort">
    <Set state="False"></Set>
</Step>
```

On (allow abort):

```
Allow User Abort [On]
```

```
<Step enable="True" id="85" name="Allow User Abort">
    <Set state="True"></Set>
</Step>
```

Most scripts that run without user interaction should set this to Off.

Set Error Capture (Step ID: 86)

On:

```
Set Error Capture [On]
```

```
<Step enable="True" id="86" name="Set Error Capture">
    <Set state="True"></Set>
</Step>
```

Off:

```
Set Error Capture [Off]
```

```
<Step enable="True" id="86" name="Set Error Capture">
    <Set state="False"></Set>
</Step>
```

`Set state="True"` = On (suppress FileMaker error dialogs; catch errors with `Get(LastError)`).

Comment (Step ID: 89)

With text:

```
# This is a comment.
```

```
<Step enable="True" id="89" name="# (comment)">
    <Text>This is a comment.</Text>
</Step>
```

Empty line (visual spacing):

```
#
```

```
<Step enable="True" id="89" name="# (comment)"></Step>
```

Comment text goes inside `<Text>` as plain text — not wrapped in CDATA.

Show Custom Dialog (Step ID: 87)

Single button:

```
Show Custom Dialog ["Notice"]
```

```
<Step enable="True" id="87" name="Show Custom Dialog">
    <Title><Calculation><![CDATA["Notice"]]></Calculation></Title>
    <Message><Calculation><![CDATA["The process is complete."]]></Calculation>
</Message>
    <Buttons>
        <Button CommitState="True"><Calculation><![CDATA["OK"]]></Calculation>
</Button>
    </Buttons>
</Step>
```

Two buttons:

```
Show Custom Dialog ["Confirm"]
```

```
<Step enable="True" id="87" name="Show Custom Dialog">
    <Title><Calculation><![CDATA["Confirm"]]></Calculation></Title>
    <Message><Calculation><![CDATA["Are you sure you want to delete this reco
rd?"]]></Calculation></Message>
    <Buttons>
        <Button CommitState="True"><Calculation><![CDATA["Delete"]]></Calcula
tion></Button>
        <Button CommitState="False"><Calculation><![CDATA["Cancel"]]></Calcul
ation></Button>
    </Buttons>
</Step>
```

Three buttons:

```
Show Custom Dialog ["Save Changes?"]
```

```
<Step enable="True" id="87" name="Show Custom Dialog">
    <Title><Calculation><![CDATA["Save Changes?"]]></Calculation></Title>
    <Message><Calculation><![CDATA["Do you want to save before closing?"]]></
Calculation></Message>
    <Buttons>
        <Button CommitState="True"><Calculation><![CDATA["Save"]]></Calculati
on></Button>
        <Button CommitState="False"><Calculation><![CDATA["Don't Save"]]></Ca
lculation></Button>
        <Button CommitState="False"><Calculation><![CDATA["Cancel"]]></Calcul
ation></Button>
    </Buttons>
</Step>
```

- Up to three `<Button>` elements are supported
- Button 1 is the default (highlighted) button
- `CommitState="True"` commits the current record when that button is clicked
- Title and Message support dynamic calculations — a literal string in quotes, or any expression

---

Appendix B — Comprehensive Step ID Catalog

All FileMaker script steps, listed alphabetically with their Step IDs. Steps marked with an asterisk (*) are covered with full examples in Appendix A.

Step Name	Step ID
# (comment) *	89
Add Account	134
Adjust Window	31
Allow Formatting Bar	115
Allow User Abort *	85
Arrange All Windows	120
AVPlayer Play	177
AVPlayer Set Options	179
AVPlayer Set Playback State	178
Beep	93
Change Password	83
Check Found Set	20
Check Record	19
Check Selection	18
Clear	49
Close Data File	196
Close File	34
Close Popover	169
Close Window	121
Commit Records/Requests *	75
Commit Transaction	206
Configure AI Account	212
Configure Local Notification	187
Configure Machine Learning Model	202
Configure NFC Reading	201
Configure Prompt Template	226
Configure RAG Account	227
Configure Region Monitor Script	185
Configure Regression Model	222
Constrain Found Set	126

Step Name	Step ID
Convert File	139
Copy	47
Copy All Records/Requests	98
Copy Record/Request	101
Correct Word	106
Create Data File	190
Cut	46
Delete Account	135
Delete All Records	10
Delete File	197
Delete Portal Row	104
Delete Record/Request *	9
Dial Phone	65
Duplicate Record/Request	8
Edit User Dictionary	109
Else *	69
Else If *	125
Enable Account	137
Enable Touch Keyboard	174
End If *	70
End Loop *	73
Enter Browse Mode	55
Enter Find Mode	22
Enter Preview Mode	41
Execute FileMaker Data API	203
Execute SQL	117
Exit Application	44
Exit Loop If *	72
Exit Script *	103
Export Field Contents	132
Export Records	36
Extend Found Set	127
Find Matching Records	155
Fine-Tune Model	213

Step Name	Step ID
Flush Cache to Disk	102
Freeze Window	79
Generate Response from Model	220
Get Data File Position	194
Get File Exists	188
Get File Size	189
Get Folder Path	181
Go to Field *	17
Go to Layout *	6
Go to List of Records	228
Go to Next Field	4
Go to Object	145
Go to Portal Row	99
Go to Previous Field	5
Go to Record/Request/Page *	16
Go to Related Record	74
Halt Script	90
If *	68
Import Records	35
Insert Audio/Video	159
Insert Calculated Result	77
Insert Current Date	13
Insert Current Time	14
Insert Current User Name	60
Insert Embedding	215
Insert Embedding in Found Set	216
Insert File	131
Insert from Device	161
Insert from Index	11
Insert from Last Visited	12
Insert from URL	160
Insert PDF	158
Insert Picture	56
Insert Text	61

Step Name	Step ID
Install Menu Set	142
Install OnTimer Script	148
Install Plug-In File	157
Loop *	71
Modify Last Find	24
Move/Resize Window	119
New File	82
New Record/Request *	7
New Window	122
Omit Multiple Records	26
Omit Record	25
Open Data File	191
Open Edit Saved Finds	149
Open Favorites	183
Open File	33
Open File Options	114
Open Find/Replace	129
Open Help	32
Open Hosts	118
Open Manage Containers	156
Open Manage Data Sources	140
Open Manage Database	38
Open Manage Layouts	151
Open Manage Themes	165
Open Manage Value Lists	112
Open Record/Request	133
Open Script Workspace	88
Open Settings	105
Open Sharing	113
Open Transaction	205
Open Upload to Host	172
Open URL	111
Paste	48
Pause/Resume Script	62

Step Name	Step ID
Perform AppleScript	67
Perform Find	28
Perform Find by Natural Language	221
Perform Find/Replace	128
Perform JavaScript in Web Viewer	175
Perform Quick Find	150
Perform RAG Action	219
Perform Script *	1
Perform Script on Server *	164
Perform Script on Server with Callback	210
Perform Semantic Find	218
Perform SQL Query by Natural Language	214
Print	43
Print Setup	42
Re-Login	138
Read from Data File	193
Recover File	95
Refresh Object	167
Refresh Portal	180
Refresh Window	80
Relookup Field Contents	40
Rename File	199
Replace Field Contents	91
Reset Account Password	136
Revert Record/Request	51
Revert Transaction	207
Save a Copy as	37
Save a Copy as Add-on Package	96
Save a Copy as XML	3
Save Records as Excel	143
Save Records as JSONL	225
Save Records as PDF	144
Save Records as Snapshot Link	152
Scroll Window	81

Step Name	Step ID
Select All	50
Select Dictionaries	108
Select Window	123
Send DDE Execute	64
Send Event	57
Send Mail	63
Set AI Call Logging	217
Set Data File Position	195
Set Dictionary	209
Set Error Capture *	86
Set Error Logging	200
Set Field *	76
Set Field By Name *	147
Set Layout Object Animation	168
Set Multi-User	84
Set Next Serial Value	116
Set Revert Transaction on Error	223
Set Selection	130
Set Session Identifier	208
Set Use System Formats	94
Set Variable *	141
Set Web Viewer	146
Set Window Title	124
Set Zoom Level	97
Show All Records	23
Show Custom Dialog *	87
Show Omitted Only	27
Show/Hide Menubar	166
Show/Hide Text Ruler	92
Show/Hide Toolbars	29
Sort Records	39
Sort Records by Field	154
Speak	66
Spelling Options	107

Step Name	Step ID
Truncate Table	182
Undo/Redo	45
Unsort Records	21
View As	30
Write to Data File	192