Suppressible Triggered Scripts

To exert more rigorous control, it can be helpful for many scripts to actively suppress the behavior of any script triggers that may be set off. This makes the behavior of a script more predictable, since triggered scripts would be known to have no effect while the main script is running. The main script could also disable and enable triggered scripts at will, selectively suppressing or taking advantage of triggered behaviors. This is accomplished with two scripting patterns and four custom functions.

Script Patterns

Any script written to be called by a trigger should begin with an If [] script step checking whether or not script triggers are currently being suppressed (after any header comments) using the TriggersAreActive custom function. That If [] statement should contain all other steps within the script, so that, in effect, the script does nothing while triggers are suppressed.

```plaintext
If [ TriggersAreActive ]
    # The rest of the triggered script goes here...
End If
```

Any script that may be disrupted by setting off script triggers can use Set Variable [] script steps to invoke the TriggersDisable and TriggersEnable custom functions as needed. These two functions should always be used in pairs.

```plaintext
Set Variable [ $ignoreMe ; TriggersDisable ]
    # Perform operations without complications from triggered scripts...
Set Variable [ $ignoreMe ; TriggersEnable ]
```

Custom Functions

TriggersAreActive

```plaintext
/**
 * =====================================
 * TriggersAreActive
 * Purpose:
 * TriggersAreActive checks a global variable to identify whether or not script
 * triggers should be allowed to run. This function is best used in concert with
 * the TriggersDisable, TriggersEnable, and TriggersReset
 * functions.
 * * Returns:
 * False (0) if triggers should be suppressed; True (1) otherwise.
 * * Parameters: none
 * =====================================
 */
```

TriggersDisable
TriggersDisable

/**
 *====================================
 * TriggersDisable
 * PURPOSE:
 * TriggersDisable sets global variables to indicate that the current
 * script is suppressing triggers. This function must be called from the context
 * of a script; otherwise, it will not suppress triggers, and it will return
 * False. In order to be suppressed, a script called via trigger should use the
 * TriggersAreActive function to decide whether or not to run.
 * RETURNS:
 * True (1) if called from the context of a script, and therefore triggers have
 * been disabled; False (0) otherwise.
 * PARAMETERS: none
 *====================================
 */

TriggersEnable

/**
 *====================================
 * TriggersEnable
 * PURPOSE:
 * TriggersEnable sets global variables to indicate that the current
 * script is no longer suppressing triggers. This function will also turn off
 * trigger suppression altogether if there are no other scripts suppressing
 * triggers. In order to be suppressed, a script called via trigger should use
 * the TriggersAreActive function to decide whether or not to run.
 * RETURNS:
 * True (1) if TriggersEnable successfully removed the current script from
 * the list of scripts suppressing triggers; False (0) otherwise.
 * PARAMETERS: none
 *====================================
 */

TriggersReset

/**
 *====================================
 * TriggersReset
 * PURPOSE:
 * TriggersReset sets global variables to indicate that script triggers
 * should be allowed to run, regardless of any persisting instructions to the
 * contrary from any script. This may be useful to recover from a situation
 * where a script forgot to call TriggersEnable at the end of operation.
 * In order to be suppressed, a script called via trigger should use the
 * TriggersAreActive function to decide whether or not to run.
 * RETURNS: ** (null)
 * PARAMETERS: none
 *====================================
 */
Sample implementations of these functions are available at the fmpstandards Github repository:

- TriggersAreActive
- TriggersDisable
- TriggersEnable
- TriggersReset

These linked functions are presented here only as examples. Different implementations may be used to achieve the same effect.