ISXEQ2 - Iterating Effects and Handling "effectInfo" Aquisition

The EQ2 client does not store in memory details for every effect/buff on you or other actors. Rather, it waits until you request the information (e.g., when you "examine" or "hover over" an effect icon), then it stores it in a cache for quick access thereafter. Therefore, ISXEQ2 only has very basic information about all of the effect/buffs (the "effect", "maintained", and "actoreffect" datatypes) until the details are requested from the server (which becomes the "effectinfo" datatype.)

The challenge for scripting is that the request for details from the server does take a couple hundred milliseconds, and ISXEQ2 can't just "sleep" and wait for that data to arrive. Therefore, when a script needs to access member(s) of the "effectinfo" datatype, it must check to see if it's available and then "wait" until it's cached.

For example:

```
function main()
{
    variable index:effect MyEffects
    variable iterator MyEffectsIterator
    variable int Counter = 1

    ;;;;;;;;;;;;;;
    ;; In this sample script, we will be using the .ToExamineInfo MEMBER of all the character's
    ;; effects, and the current target's effects.   The routines below will properly handle
waiting
    ;; until the effectinfo object is cached and available; however, since there is only ever a
maximum
    ;; of 30 beneficial, 30 detrimental, and 30 actor effects available to the client at any given
time,
    ;; ISXEQ2 provides a useful method to start the process of getting the details cached for all
effects
    ;; and buffs at once.   This should save time in the iteration routines below.
    ;;;;;;;;;;;;;;
    Me:RequestEffectsInfo
    if (${Target(exists)})
        Target:RequestEffectsInfo

    ;;;;;;;;;;;;;;
    ;; Beneficial Effects
    ;;;;;;;;
    ;;;;
    ;;;; The following routine illustrates how to iterate through beneficial effects.  To return a
single
    ;;;; effect, you can do so by using the "Query" argument along with a lavishsoft Query String.
 For example
    ;;;; to check if the character has a beneficial effect with a MainIconID of 234, simply use:
    ;;;; "if ${Me.Effect[Query, Type == "Beneficial" && MainIconID == "234"](Exists)}"
    ;;;;;;;;

    echo "Beneficial Effects:"
    Me:QueryEffects[MyEffects, Type == "Beneficial"]
    MyEffects:GetIterator[MyEffectsIterator]

    if ${MyEffectsIterator:First(exists)}
    {
        do
        {
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;; This routine is echoing the effect's "Name", so we must ensure that the effectinfo
            ;; datatype is available.
            if (!${MyEffectsIterator.Value.IsEffectInfoAvailable})
            {
                ;; When you check to see if "IsEffectInfoAvailable", ISXEQ2 checks to see if it's
already
                ;; cached (and immediately returns true if so).  Otherwise, it spawns a new
thread
                ;; to request the details from the server.
                do
```

```
                {
                    waitframe
                    ;; It is OK to use waitframe here because the "IsEffectInfoAvailable" will
simple return
                    ;; FALSE while the details acquisition thread is still running.   In other
words, it
                    ;; will not spam the server, or anything like that.
                }
                while (!${MyEffectsIterator.Value.IsEffectInfoAvailable})
            }
            ;;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;; At this point, the "ToEffectInfo" MEMBER of this object will be immediately
available.  It should
            ;; remain available until the cache is cleared/reset (which is not very often.)

            echo "- ${Counter}. ${MyEffectsIterator.Value.ToEffectInfo.Name} (ID:
${MyEffectsIterator.Value.ID}, MainIconID: ${MyEffectsIterator.Value.MainIconID}, BackDropIconID:
${MyEffectsIterator.Value.BackDropIconID})"
            Counter:Inc
        }
        while ${MyEffectsIterator:Next(exists)}
    }
    else
        echo "NONE"
    echo "----------------------"
    Counter:Set[1]
    ;;
    ;;;;;;;;;;;;;;

    ;;;;;;;;;;;;;;
    ;; Detrimental Effects
    ;;;;;;;;
    ;;;;
    ;;;; The following routine illustrates how to iterate through detrimental effects.  To return
a single
    ;;;; effect, you can do so by using the "Query" argument along with a lavishsoft Query String.
 For example
    ;;;; to check if the character has a detrimental effect with a MainIconID of 234, simply use:
    ;;;; "if ${Me.Effect[Query, Type == "Detrimental" && MainIconID == "234"](Exists)}"
    ;;;;;;;;

    echo "Detrimental Effects:"
    Me:QueryEffects[MyEffects, Type == "Detrimental"]
    MyEffects:GetIterator[MyEffectsIterator]

    if ${MyEffectsIterator:First(exists)}
    {
        do
        {
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;; This routine is echoing the effect's "Name", so we must ensure that the effectinfo
            ;; datatype is available.
            if (!${MyEffectsIterator.Value.IsEffectInfoAvailable})
            {
                ;; When you check to see if "IsEffectInfoAvailable", ISXEQ2 checks to see if it's
already
                ;; cached (and immediately returns true if so).  Otherwise, it spawns a new
thread
                ;; to request the details from the server.
                do
                {
                    waitframe
                    ;; It is OK to use waitframe here because the "IsEffectInfoAvailable" will
simple return
                    ;; FALSE while the details acquisition thread is still running.   In other
words, it
```

```
                    ;; will not spam the server, or anything like that.
                }
                while (!${MyEffectsIterator.Value.IsEffectInfoAvailable})
            }
            ;;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;; At this point, the "ToEffectInfo" MEMBER of this object will be immediately
available.  It should
            ;; remain available until the cache is cleared/reset (which is not very often.)

            echo "- ${Counter}. ${MyEffectsIterator.Value.ToEffectInfo.Name} (ID:
${MyEffectsIterator.Value.ID}, MainIconID: ${MyEffectsIterator.Value.MainIconID}, BackDropIconID:
${MyEffectsIterator.Value.BackDropIconID})"
            Counter:Inc
        }
        while ${MyEffectsIterator:Next(exists)}
    }
    else
        echo "NONE"
    echo "---------------------"
    Counter:Set[1]
    ;;
    ;;;;;;;;;;;;;;


    ;;;;;;;;;;;;;;
    ;; Maintained Effects (Buffs)
    ;;;;;;;;
    ;;;;
    ;;;; The following routine illustrates how to iterate through maintained effects (buffs).  To
return a single
    ;;;; buff, you can do so by searching for a particular buff name or the index of the buff in
the array (i.e.,
    ;;;; ${Me.Maintained["NAME"]} or ${Me.Maintained[#]})
    ;;;;;;;;

    echo "Maintained Effects (Buffs):"
    variable string Duration
    variable string EffectTarget
    variable int CountMaintained

    CountMaintained:Set[${Me.CountMaintained}]

    if (${CountMaintained} > 0)
     {
         do
         {
            if (${Me.Maintained[${Counter}].Duration.Equal[-1]})
                Duration:Set["N/A"]
            else
                Duration:Set[${Me.Maintained[${Counter}].Duration.Precision[2]}]

            if (${Me.Maintained[${Counter}].Target(exists)})
                EffectTarget:Set[${Me.Maintained[${Counter}].Target.Name}]
            else
                EffectTarget:Set["N/A"]

            echo "- ${Counter}. ${Me.Maintained[${Counter}].Name} (Duration: ${Duration}, Target:
${EffectTarget})"

         }
         while (${Counter:Inc} <= ${CountMaintained})
     }
    else
        echo "NONE"
    echo "---------------------"
    Counter:Set[1]
```

```
    ;;
    ;;;;;;;;;;;;;;

    ;;;;;;;;;;;;;;
    ;; ActorEffects (Current Target)
    ;;;;;;;;
    ;;;;
    ;;;; The following routine illustrates how to iterate through actor effects.  To return a
single
    ;;;; actor effect, you can do so by using the "Query" argument along with a lavishsoft Query
    ;;;; string.   For example, to check if the current target has the effect with an ID of
123456, one could
    ;;;; check "if ${Target.Effect[Query,ID = "123456"](Exists)}"
    ;;;;;;;;

    echo "ActorEffects on the current target:"
    variable int NumActorEffects = 0

    if (${Target(exists)})
    {
        NumActorEffects:Set[${Target.NumEffects}]

        if (${NumActorEffects} > 0)
        {
            do
            {

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;; This routine is echoing the effect's "Name", so we must ensure that the
effectinfo
                ;; datatype is available.
                if (!${Target.Effect[${Counter}].IsEffectInfoAvailable})
                {
                    ;; When you check to see if "IsEffectInfoAvailable", ISXEQ2 checks to see if
it's already
                    ;; cached (and immediately returns true if so).  Otherwise, it spawns a new
thread
                    ;; to request the details from the server.
                    do
                    {
                        waitframe
                        ;; It is OK to use waitframe here because the "IsEffectInfoAvailable" will
simple return
                        ;; FALSE while the details acquisition thread is still running.   In other
words, it
                        ;; will not spam the server, or anything like that.
                    }
                    while (!${Target.Effect[${Counter}].IsEffectInfoAvailable})
                }
                ;;

 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                ;; At this point, the "ToEffectInfo" MEMBER of this object will be immediately
available.  It should
                ;; remain available until the cache is cleared/reset (which is not very often.)

                echo "- ${Counter}. ${Target.Effect[${Counter}].ToEffectInfo.Name} (ID:
${Target.Effect[${Counter}].ID}, MainIconID: ${Target.Effect[${Counter}].MainIconID})"
            }
            while (${Counter:Inc} <= ${NumActorEffects})
        }
        else
            echo "NONE"
    }
    else
        echo "NO TARGET"
    echo "---------------------"
```

```
    ;;
    ;;;;;;;;;;;;;;;
}
```