ISXEQ2 - Iterating Abilities and Handling "abilityInfo" Aquisition

The EQ2 client does not store in memory details for every character ability. Rather, it waits until you request the information (e.g., when you "examine" an ability), then it stores it in a cache for quick access thereafter. Therefore, ISXEQ2 only has very basic information about abilities (the "ability" datatype) until the details are requested from the server (which becomes the "abilityinfo" 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 "abilityinfo" datatype, it must check to see if it's available and then "wait" until it's cached.

In the example below, you will need to change line 17 to utilize one or more of the ability IDs that your current character possesses. Otherwise, the script will return NONE. (Iterating all abilities is possible; however, it does take a while on higher level characters.)

Example

```
function main()
{
    variable index:ability MyAbilities
    variable iterator MyAbilitiesIterator
    variable int Counter = 1
    variable int Timer = 0

    ;;;;;;;;;;;;;;
    ;;;;
    ;;;; The following routine illustrates how to iterate through abilities.  To return a single
    ;;;; ability, you can do so by using the "Query" argument along with a lavishsoft Query
String.  For example
    ;;;; to check if the character has an ability with a ID of 234, simply use:
    ;;;; "if ${Me.Ability[Query, ID =- "546331599"](Exists)}"
    ;;;;;;;;

    echo "Abilities (Total ${Me.NumAbilities}):"
    Me:QueryAbilities[MyAbilities, ID =- "546331599" || ID =- "478256501"]
    MyAbilities:GetIterator[MyAbilitiesIterator]

    if ${MyAbilitiesIterator:First(exists)}
    {
        do
        {
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            ;; This routine is echoing the ability's "Name", so we must ensure that the
abilityinfo
            ;; datatype is available.
            if (!${MyAbilitiesIterator.Value.IsAbilityInfoAvailable})
            {
                ;; When you check to see if "IsAbilityInfoAvailable", 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
                {
                    wait 2
                    ;; It is OK to use waitframe here because the "IsAbilityInfoAvailable" 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 (!${MyAbilitiesIterator.Value.IsAbilityInfoAvailable} && ${Timer:Inc} <
1500)
            }
            ;;
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
            ;; At this point, the "ToAbilityInfo" 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}. ${MyAbilitiesIterator.Value.ToAbilityInfo.Name} (ID:
${MyAbilitiesIterator.Value.ID}, IsReady: ${MyAbilitiesIterator.Value.IsReady})"
            Counter:Inc
            Timer:Set[0]
        }
        while ${MyAbilitiesIterator:Next(exists)}
    }
    else
        echo "NONE"
    echo "----------------------"
    ;;
    ;;;;;;;;;;;;;;
}
```