BrightSign®

TECHNICAL NOTES

Using Plugins and Parsers with BrightAuthor

BrightSign, LLC. 16795 Lark Ave., Suite 200 Los Gatos, CA 95032
408-852-9263 | www.brightsign.biz

INTRODUCTION

This tech note describes how to use two advanced BrightAuthor features: custom autorun plugins and parser scripts. These instructions assume a certain level of familiarity with BrightScript and coding practices. This is not a comprehensive guide to writing custom scripts for BrightAuthor; rather, it is meant to provide a general outline and best practices for writing those scripts.

Custom Autorun Plugins

Custom plug-ins allow you to add script extensions to a standard presentation autorun. Plugins have two primary benefits over custom autoruns: They can be easily inserted into multiple presentations, including newer and older versions of the same presentation; and they are not dependant on a certain autorun or firmware version, greatly reducing the complexity involved in updating custom BrightScript deployments.
Note: *Custom plugins are only available in BrightAuthor 3.7 or later.*

To designate one or more custom plugins for a BrightAuthor presentation, navigate to **File > Presentation Properties > Autorun**. Click the **Add Script Plugin** button, give the plugin a name, and click the **Browse** button to locate and select a *.brs* file to use as a plugin.

Plug-in scripts must include an initialization function in the form of `<plugin_name>_Initialize`. This initialization function is passed three parameters:

- `msgPort as Object`
- `userVariables as Object`
- `o as Object`: This is the bsp associative array from the autorun. It is required for the initialization function to return an associative array.

To process events, the plugin script must provide a `ProcessEvent` function that is a member of the associative array returned by the initialization function described above. The

`ProcessEvent` function is given a single event object. The function then returns a Boolean indicating whether or not the autorun should continue processing the event object: The function will return True if it handles the event and the standard autorun should not continue processing the event.

The associative array that defines the object must also include an `objectName` entry that defines the name of the object.

The following is an example of a script plugin file named “pizza.brs”:

```
Function pizza_Initialize(msgPort As Object, userVariables As Object, o As
Object)

    print "pizza_Initialize - entry"
    print "type of msgPort is ";type(msgPort)
    print "type of userVariables is ";type(userVariables)

    PizzaBuilder = newPizzaBuilder(msgPort, userVariables)

    return PizzaBuilder

End Function

Function newPizzaBuilder(msgPort As Object, userVariables As Object)

    PizzaBuilder = { }
    PizzaBuilder.msgPort = msgPort
    PizzaBuilder.userVariables = userVariables
    PizzaBuilder.objectName = "PizzaBuilder_object"

    PizzaBuilder.ProcessEvent = pizza_ProcessEvent
```

```
    return PizzaBuilder


End Function


Function pizza_ProcessEvent(event As Object)


    print "pizza_ProcessEvent - entry"
    print "type of m is ";type(m)
    print "type of event is ";type(event)


' swallows timer events - telling the autorun not to process them
    if type(event)= "roTimerEvent" then
       return true
    else
       return false
    endif


End Function
```

Receiving a Plugin Message

The following example code shows how to write a script that receives a **Plugin Message Command** from the autorun. This code listens for a message sent to the plugin named “Pizza” and then prints the message:

```
Function pizza_ProcessEvent(event As Object)


    print "pizza_ProcessEvent - entry"
    print "type of m is ";type(m)
    print "type of event is ";type(event)
```

```
    if type(event) = "roAssociativeArray" then
        if type(event["EventType"]) = "roString"
            if event["EventType"] = "SEND_PLUGIN_MESSAGE" then
                if event["PluginName"] = "Pizza" then
                    pluginMessage$ = event["PluginMessage"]
                    print "received pluginMessage ";pluginMessage$
                    return true
                endif
            endif
        endif
    endif

    return false

End Function
```

Sending a Plugin Message

The following example code shows how to write a script that sends a message string to trigger a **Plugin Message** event.

```
print "pizza_ProcessEvent - entry"
    print "type of m is ";type(m)
    print "type of event is ";type(event)

    if type(event) = "roTimer" then

        pluginMessageCmd = CreateObject("roAssociativeArray")
        pluginMessageCmd["EventType"] = "EVENT_PLUGIN_MESSAGE"
        pluginMessageCmd["PluginName"] = "Pizza"
        pluginMessageCmd["PluginMessage"] = "toppings"
```

```
        m.msgPort.PostMessage(pluginMessageCmd)
        return true

    endif

    return false

End Function
```

Parser Scripts

Parser scripts allow you to manipulate data sets from incoming RSS feeds. You can create parser scripts for a wide array of system and presentation functions. You can designate a parser for an RSS feed by navigating to **File > Presentation Properties > Data Feeds** in BrightAuthor.
Note: *The parser file must have a* .brs *extension.*

The following parameters can be used with the parser subroutine:

- `xmlFileName$`: The name of the XML file. This information is provided by the autorun.
- `itemsByIndex`: An array of the descriptions. The script should fill in this array if the Live Text object is using the “item index” to display items from this feed.
- `itemsByTitle`: An associative array of titles, descriptions. The script should fill in this array if a Live Text object is using the “item title” to display items from this feed.
- `userVariables`: An associative array of current User Variables (which are provided by the autorun). This is provided in case you want to analyze or modify the User Variables.

The following piece of example code can be used to parse RSS text:

```
       xml = CreateObject("roXMLElement")
       if not xml.Parse(ReadAsciiFile(xmlFileName$)) then
               print "xml read failed"
       else
               if type(xml.channel.item) = "roXMLList" then
```

```
                        index% = 0

                        for each itemXML in xml.channel.item

                                itemsByIndex.push(stri(index%)     +     "     -     "     +
itemXML.description.GetText())

                                index% = index% + 1

                        next

                endif

        endif


end Sub
```