

There is no Spoon



Machineri Nirvana

A Guide through the Galaxy by Marvin Siddharta

Index

Find the name of the chapter you'd wish to read up on here, before diving into the 'big pie'
...

1. Welcome to Magix!

A short introduction to Magix and what you can expect from it. Explains the terms and introduces Magix using a Tutorial-Like communication form. This is the main content parts of the book, while the other parts are more reference like in structure. Start out here to get an introduction of the different terms if you are new to Magix, and use the other parts as a Reference Guide later when wondering about specific subjects. Also remember that there are tons of stuff about Magix on YouTube and on the internet in general in case you're stuck with a very specific subject

2. Meta Actions

Reference documentation about the MetaActions that was present, and had documentation within your installation upon the generation of this document

3. CompressionModule

Level3: Helper class for implementing GZIP-Compression of resources and pages. PS! Thanx and credits goes to Mads Kristiansen for sharing this on his blog. Google 'CompressionModule Mads Kristiansen' if more background information is needed about this piece of code

4. MainWebPage

Level4: Your 'Application Pool', meaning the 'world' where all your 'components' lives. Nothing really to see here, this should just 'work'. But are here for reference reasons

5. Magix_PageStatePersister

Level4: Implements serialization of ViewState into the database on the server side

6. ActiveTypeCached-g

Level3: Helper class for going through cache to retrieve Active Types. Will cache most queries

7. TextAreaEdit

Level4: Basically every single instance you see in Magix which is of type 'InPlaceEditable' is an instance of this component. Allows for viewing textual based information, while at the same time, this tet can be 'clicked' which will change it into a text box, in which you can change its value. Often displayed with a pen next to it

8. Module

Level4: Baseclass for the 'Edit Objects' parts of the Grid system in Magix. Not meant for directly consuming through this class

9. ListModule

Level4: Implements most of the logic for our DBAdmin module and basically every single Grid we have in Magix. This class is the foundation for probably most Magix screens you've ever seen, since every Grid within the system, virtually, is built using this class. Have so many bells and whistles, it could probably need its own book, not to mention some serious refactoring. Till at least the refactoring parts are over, I think I'll resist the temptation of going 'over the board' in documenting this, since first of all everything will change later, and you shouldn't fiddle too much directly with this bugger yourself. There's also PLENTY code samples around ANYWHERE in magix of usage of this class, so have fun :)

10. ClipBoard_Controller

Level2: Encapsulates the logic needed for manipulating the clipboard, add items to it, and paste items to the Clipboard. Notice that by overriding some of these events, and have them go to other servers, you can copy and paste any clipboard items from one server to the other

11. Common_Controller

Level2: Contains some common helpers and logic that doesn't belong at one particular place. For instance loading of a color picker dialogue

12. DBAdmin_Controller

Level2: Contains the logic for the DBAdmin module(s), which is also the Grid/CRUD-foundation system in Magix. Contains many useful methods and ActiveEvents for displaying either Grids or editing Single Instances of Objects. Can react 100% transparently on ActiveTypes. Has support for Meta Types, meaning types where you're more in 'control', but must do more of the 'arm wrestling directly'

13. ColumnTypesPlugins_Controller

Level3: Contains some template columns for the Grid system for you to use in your own Grids

14. Distributor_Controller

Level2: Helper class for serializing and transmitting back and forth between other servers

15. Documentation_Controller

Level2: Contains the logic for our 'Class Browser' which can browse all the classes in the system and make some changes to them by enabling them and disabling them by either overriding specific events or by disabling entire controllers or modules all together

16. Email_Controller

Level2: Implements logic for sending email using SMTP through the .Net classes

17. FileExplorer_Controller

Level2: Contains Logic for file explorer. The file explorer is a component where you can browse the file folder system on your web server. Such as editing CSS files, uploading images and such. PS! To use this Module it is IMPERATIVE that you've given the 'NETWORK SERVICE' account 'Full Access' to at the very least the 'media/' folder, or whatever folder you plan to use the explorer at on your web server

18. GalleryPlugin_Controller

Level2: Gallery plugin for the Gallery modules. Also serves as a nice vantage point for creating your own plugins to Magix and the publishing system

19. Logger_Controller

Level2: Contains logic for Logging things

20. Games_Controller

Level2: Contains logic to load up games in menu etc

21. DataBinding101_Controller

Level2: Will upon startup create an Image Button which 'flashes' which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

22. MetaForms_Controller

Level2: Contains logic to help build Meta Forms. Meta Forms are forms where you can almost in its entirety decide everything about how a View can be built

23. CreateLoginForm_Controller

Level2: Will upon startup create a Login Form which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

24. CreateHelloWorldForm_Controller

Level2: Will upon startup create a Login Form which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

25. CreateFlashingImageButton_Controller

Level2: Will upon startup create an Image Button which 'flashes' which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

26. MetaObject_Controller

Level2: Contains logic for editing, maintaining and viewing MetaObjects. MetaObjects are at the heart of the Meta Application System since they serve as the 'storage' for everything a view updates or creates through interaction with the end user

27. UploaderPlugin_Controller

Level2: Contains logic for being able to upload files to the server

28. MetaAction_Controller

Level2: Contains logic for Actions which are wrappers around ActiveEvents for the end user to be able to raise his own events, with his own data in the Node structure. The MetaAction system is at the core of the Meta Application system wince without it the end user cannot create his own types of events or Actions

29. CommonActions_Controller

Level2: Contains common end user useful actions which doesn't really belong any particular place, but which can still be immensely useful for scripting purposes. Perceive these as pugins, or extra candy if you wish. Often they're simplifications of other more hard core Active Events. This is one of the most interesting classes for you to look at in order to figure out what events to start using

30. CreateDefaultInitialActions_Controller

Level2: Contains Application Startup code to create the defaul Actions unless they're already there

31. MetaView_Controller

Level2: Contains helper logic for viewing and maintaing MetaViews, and related subjects. MetaViews are the foundation for the whole viewing parts of the Meta Application system. A MetaView is imperativ for both being able to collect new data and also for viewing existing data. The MetaView defines which parts of the object you can see at any time too, which means you can use it to filter access according to which Grid the user is having access to, for instance. This controller contains logic for editing and maintaining MetaViews, plus also direct usage of MetaViews

32. PDF_Controller

Level2: Contains logic for creating PDF files and downloading to Client or saving locally

33. Logging_Controller

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

34. LoginOut_Controller

Level2: Login and Logout Controller

35. FileSystem_Controller

Level2: Tiny helper for menu item in Administrator Dashboard to view the File System Browser

36. InitialLoading_Controller

Level2: Class for taking care of the Magix.Core.InitialLoading message, meaning the initialization of the page on a per User level initially as they're coming to a new 'page' or URL ... Basically just handles Magix.Core.InitialLoading and mostly delegates from that point ...

37. SliderMenu_Controller

Level2: Helps feed the SliderMenu in Front-Web to get its Items to build its structure upon

38. TipOfToday_Controller

Level2: Creates our default Tooltips or Tutorial rather

39. OpenID_Controller

Level2: Controller implementing the OpenID logic for both Relying Party and OpenID Provider logic. Every User in Magix can use the website where Magix is installed as an OpenID provider by typing in the root URL of the web application and append ?openID=username In addition you can associate as many OpenID tokens as you wish to your User and then use these other OpenID providers to log into Magix.

40. PageLoader_Controller

Level2: Controller responsible for loading up 'Page Objects', and doing initialization around pages and such as they're being loaded

41. WebPart_Controller

Level2: Initializes WebParts and such while being injected onto WebPage

42. AdministratorDashboard_Controller

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

43. ChildExcerpt_Controller

Level2: Helps out with some of the editing and using of the ChildExcerpt Module Type

44. EditUser_Controller

Level2: Here mostly for Editing User, and also to some extent creating default users and roles upon startup if none exists

45. AdminDashboardHeader_Controller

Level2: We'll 'lock' the Header control since it can be VERY annoying sometimes due to the DB Manager, which seriously needs to be refactored here BTW ...

46. EditRoles_Controller

Level2: Here mostly to serve up grids and such for editing of Roles in the system

47. EditTemplates_Controller

Level2: Controller for editing Templates. Contains all the relevant event handlers and logic for editing your templates

48. Dashboard_Controller

Level2: Main 'router' in dispatching important Dashboard functionality

49. EditPages_Controller

Level2: Contains logic for Editing WebPage objects and such for administrator

50. Access_Controller

Level3: Helps out sorting out which Pages and Menu Items which Users and Roles and such have access to

51. QRGenerator_Controller

Level2: Implements logic to create, generate and handle QR-Codes

52. Settings_Controller

Level2: Helper logic to assist with Settings and updating of settings. PS! Unless you have this controller in your Application Pool, Enabled, then direct changes, through the DB Admin module to Settings will NOT affect the Application Pool before it's being re started ...

53. Signature_Controller

Level2: Signature Plugin Control for Publishing system to allow for loading of a different plugin module from e.g. the click of a row button or a single-view button etc.

54. TalkBack_Controller

Level2: Contains helper logic for the TalkBack module [forums] in Magix

55. ToolTip_Controller

Level2: Contains helper logic for displaying and maintaining previous/next Tip of today and such

56. Clips

Level2: Will display a list of all of your ClipBoard items

57. Text

Level2: Encapsulates a Text module. Pass in 'Text' and optionally 'Tag'

58. Separator

Level2: Encapsulates a [dead] html Horizontal Ruler [hr] element for you if you need one

59. PickColorOrImage

Level2: Colorpicker Module for allowing the end user [or you] to pick colors in the system. Also has support for picking images for use as textures as alternatives to picking color

60. ViewPortContainer

Level2: Encapsulates a ViewPort Container Component, basically a container for other modules

61. Tree

Level2: Shows a Tree module for the end user for him to navigate and select single nodes from. Change its properties by passing in 'TreeCssClass' or 'NoClose'. 'Items' should contain the tree items in a hierarchical fashion with e.g. 'Item/i-1/' containing 'Name' 'CssClass' and 'ToolTip'. 'Name' being minimum. Child items of 'Items/i-1' should be stored in the 'Items/i-1/Items' node. Will raise 'GetItemsEvent' upon needing to refresh for some reasons, and 'ItemSelectedEvent' with 'SelectedItemID' parameter as selected item ID upon user selecting an item

62. TipOfToday

Level2: Shows one tip of today with the option for the User to browse forward or backward to read more. Raises 'Magix.Core.GetPreviousToolTip' and 'Magix.Core.GetNextToolTip' to get its next and previous tips

63. Header

Level2: Encapsulates a header [h1] control for you to create headers for your pages and apps. Load it and raise 'Magix.Core.SetFormCaption' to set the header

64. Clickable

Level2: Encapsulates a Clickable Component, basically a button. Which you can load as a module, and trap clicks to. Set its 'Text', 'ButtonCssClass', 'Enabled' and 'ToolTip' params to customize it till your needs. Will raise 'Event' when clicked

65. AnalogClock

Level2: Shows an Analog Clock module

66. MessageBox

Level2: Implements logic for showing the end user a message box, asking for confirmation or something similar when needed. 'Text' will be the text shown to the user, which he should read and take a stand in regards to. Dropping the 'Cancel' Node sets Cancel to invisible if you wish. 'OK/Event' will be raised with 'OK' node if OK button is clicked. 'Cancel/Event' will be raised with 'Cancel' node if Cancel button is clicked

67. ImageModule

Level2: Control for displaying images in your app. Either clickable or static images. Pass in 'ImageURL', 'AlternateText', 'ChildCssClass' and 'Description' to modify it according to your needs. Description, if given, will add a label underneath the image. Use 'Seed' to have multiple Images on same page and to separate between different instances of them. If 'Events/Click' is given, it'll raise that event upon clicking the image

68. ImageList

Level2: Modules for displaying a List of Images. Pass in your Image list in the 'Items' parameter to the LoadModule method

69. Uploader

Level3: ActiveModule to help create upload files logic. Will raise 'Magix.Uploader.FileUploaded' consecutively for every file uploaded, and 'Magix.Uploader.FileBatchUploadFinished' when all files in one drag and drop operation are done. Alternatively it may raise the event defined in 'FileUploadedEvent' or 'AllFilesUploadedEvent'

70. ViewClassContents

Level2: Will show all objects of type 'FullTypeName' in a Grid from which the user can filter, edit, change values, delete objects, create new objects and such from. Basically the 'show single table' logic of the Magix 'Database Enterprise Manager'. If 'IsCreate', will allow for creation of objects of the 'FullTypeName' by the click of a button

71. ViewListOfObjects

Level2: Basically the same as ViewClassContents, though will only show objects 'belonging to a specific object [ParentID] through a specific property [PropertyName]' and allow for appending, and not creation of new objects of 'FullTypeName'. Raises 'DBAdmin.Data.GetListFromObject' to get objects to display in Grid. Override the Append button's text property with 'AppendText'. Other properties are as normal from mostly every grid in Magix such as 'IsDelete', 'IsRemove' etc

72. ViewSingleObject

Level2: Contains the logic for editing and viewing one single ActiveType object, with all of its properties. Can become initiated in two different states, one of which is 'edit object reference from another object' which will allow for changing and removing the reference, the other is plain old 'edit the thing' mode. Supports 'ChildCssClass' and several other properties. Most of the common properties from the Database Enterprise Manager is included. Will raise 'DBAdmin.Form.ChangeObject' if user attempts to change the reference. Will raise 'DBAdmin.Data.RemoveObject' when object reference is removed. Changing the reference or removing it is only enabled if 'IsChange' and/or 'IsRemove' is given as true

73. FindObject

Level2: Basically identical to ViewClassContents, with one addition. This module will show a 'filter field' to the end user which the user can use to 'search for items' from within.

74. BrowseClasses

Level2: Contains UI for letting the end user browse the ActiveType classes within his system as a Database Enterprise Management tool. Allows for seeing all classes in a Tree hierarchy and letting him select classes, which again will trigger editing of records in that class. Kind of like the Database Enterprise Management tool for Magix. Will raise 'DBAdmin.Data.GetClassHierarchy', which you can handle if you have 'meta types' you wish to display

75. ConfigureColumns

Level2: Allows for editing of 'visible columns' per type level. Will show a form from which the end user can tag off and on the different columns he wish to see in his grid. Databinds towards 'WhiteListColumns' and raises 'DBAdmin.Data.ChangeVisibilityOfColumn' when visibility of column changes with 'Visible', 'FullTypeName' and 'ColumnName' as parameters

76. ConfigureFilters

Level2: Will allow the end user to configure his filters, meaning what filter to filter each column by. There are many different types of filter in Magix. LIKE, Equal and so on. This form allows you to globally set the filters for specific types on specific columns of those types

77. ShowClassDetails

Level2: Doxygen helper class for displaying documentation about members of classes and such for our documentation system. Takes in 'FullName', 'Description' and so on. Will create a grid if displaying items according to the structure given

78. RichEdit

Level2: Contains the UI for the RichEditor or WYSIWYG editor of Magix. That little guy resembling 'word'. Specify a 'SaveEvent' to trap when 'Text' is being edited.

79. Explorer

Level2: Containe the UI for the Explorer component, which allows you to browse your File System on your server, through your browser. Basically a File System Explorer kind of control, which allows for renaming, deleting, and editing [to some extent] the files in your installation. Can be instantiated in Select mode by setting its 'IsSelect' input parameter. If 'CanCreateNewCssFile' is true, the end user is allowed to create a new default CSS file which he can later edit. 'RootAccessFolder' is the root of the system from where the current user is allowed to browse, while 'Folder' is his current folder. The control does some basic paging and such, and has support for will raise 'Magix.FileExplorer.GetFilesFromFolder' to get its items. The module will raise the value of the 'SelectEvent' paeameter when an item has been selected. The module supports browsing hierarchical folder structures

80. EditAsciiFile

Level2: Kind of like Magix' version of 'Notepad'. Allows for editing of textfiles or text fragments, and allow for saving them. Pass in either 'File' [pointing to an absolute file name] or 'Text' which should be textual fragments. If 'NoSave' is True, it won't allow for saving of the document

81. Tetris

Level2: Contains the Tetris game

82. Slider

Level2: Contains the UI for our SlidingMenu module, used in the administrator dashboard. Takes a recursive 'Items' structure containing 'Caption' and 'Event' which will be raised when clicked ['Event']

83. StyleBuilder

Level2: Style builder for creating CSS style collections and such for animations, CSS classes, styles etc

84. MetaView_Form

Level2: Encapsulates a MetaForm through a PublisherPlugin which you can inject into your sites

85. EditMetaForm

Level2: Helper for editing and creating Meta Form objects

86. MetaView_Single

Level2: UI parts for showing a MetaView in 'SingleView Mode'. Basically shows a form, with items dependent upon the look of the view. This is a Publisher Plugin module. This form expects to be given a 'MetaViewName', which will serve as the foundation for raising the 'Magix.MetaView.GetViewData' event, whos default implementation will populate the node structure according to the views content in a Key/Value pair kind of relationship. This will serv as the foundation for the module to know which types of controls it needs to load up [TextBoxes, Buttons etc]Handles the 'Magix.MetaView.SerializeSingleViewForm' event, which is the foundation for creating new objects upon clicking Save buttons etc.This is the PublisherPlugin you'd use if you'd like to have the end user being able to create a new MetaObject

87. MetaView_Multiple

Level2: UI parts for showing a MetaView in 'MultiView Mode'. Basically shows a grid, with items dependent upon the look of the view. This is a Publisher Plugin module. Basically a completely 'empty' module whos only purpose is to raise the 'Magix.MetaView.ShowMetaViewMultipleInCurrentContainer' event, whos default implementation will simply in its entirety replace this Module, hence not really much to see here.This is the PublisherPlugin you'd use if you'd like to see a 'list of MetaObjects' on the page

88. ImageGallery

Level1: Represents an Image Gallery plugin for the Publishing system. Allows for browsing of Images through a Gallery lookalike UI

89. Uploader

Level2: File Uploader plugin module for the Publishing system. By having this module on your WebPage, you will allow the end user to drag and drop files onto anywhere on the browser, which will trigger uploading of those files to the server

90. Header

Level2: A 'header' PublisherPlugin. Basically just an HTML h1 element

91. EditSpecificTemplate

Level2: Allows for editing of WebPageTemplate objects. Contains most of the UI which you're probably daily using while adding and creating new templates and such

92. LogInOutUser

Level2: PublisherPlugin containing most of the UI for allowing a user to login/out of your website. Can be set into both OpenID mode and 'only native mode' or both. Raises 'Magix.Publishing.GetStateForLoginControl' to determine the state of the module, meaning if it should show both OpenID logic and native logic or only one of them. Will raise 'Magix.Core.UserLoggedOut' if user logs out and 'Magix.Core.LogInUser' if the user tries to log in. The 'Magix.Core.LogInUser' default implementation again will raise 'Magix.Core.UserLoggedIn' if it succeeds. It'll pass in 'OpenID' if user has chosen to log in with OpenID and 'Username'/'Password' if user choses to login natively

93. TopMenu

Level2: PublisherPlugin containing a conventional 'File Menu' type of menu which normally is expected to find at the top of a page, which will 'drop down' child selection boxes for being able to select children. Useful for conventional applications, which should look like legacy code, or something. Takes the exact same input parameters as the SliderMenu PublisherPlugin

94. SliderMenu

Level2: PublisherPlugin containing the UI for the Sliding Menu Publisher Plugin. Meaning the default menu to the left in the front-web parts, which you can choose to inject into a WebPartTemplate in one of your Templates if you wish. Basically just loads the items once through raising the 'Magix.Publishing.GetSliderMenuItems' event, which should return the items as a 'Items' list, containing 'Caption', 'Event', 'Event' [Event name] and 'Event/WebPageURL' which normally will contain the page's URL

95. Content

Level2: A 'content' PublisherPlugin. Basically just a text fragment, that'll be edited through the Magix' RichText or WYSIWYG Editor

96. ChildExcerpt

Level2: Will show an excerpt of all its children, sorted with newest first, showing a maximum of PagesCount items. Kind of like a front page to a blog or a news website or something

97. CreateImage

Level2: Encapsulates a module for selecting an image

98. EditSpecificPage

Level2: Allows for editing of one single specific WebPage in the system. Contains most of the UI which you're probably daily using while adding and creating new pages and such

99. CreateLink

Level2: Encapsulates a module for selecting object, image, etc to link to

100. Generator

Level2: Vanity QR Code Generator for creating 'beautiful' QR Codes. Also contains an algorithm [burn] which creates more reciliant QR codes that scans faster than conventional square dots

101. Sign

Level2: A Signature module, basically a place where the end user can 'sign his name' to confirm a transaction of some sort. Will load up a big white thing, which can be 'drawn upon', together with two buttons, OK and Cancel which will raise the 'CancelEvent' and the 'OKEvent'. 'OKEvent' will pass in 'Signature' being the coords for all the splines that comprises the Signature, which can be stored and later used as input to this Module, which will again load those splines in 'read-only mode'

102. Forum

Level2: Basically a 'Forum module' which allows for posting and reading and replying to other people's opinions about 'whatever'. Not very good on its own, please use through TalkBack Controller to save head aches

103. Login

Shows a login box with username/password/openid combo. Username/password has preference, but if only OpenID is given, the system will attempt at login you in using the OpenID claim. Note, this is not the Publisher Plugin login box. This is the 'main system one', which you get by going to ?dashboard=true

104. Settings

Level2: Wrapper class for common/global settings within a Magix-Brix application. Use the overloaded this operator to access Settings through code

105. TipOfToday

Level3: Helper class for Tip [Today's Tips] logic

106. Cookie

Level2: Encapsulates a Cookie for a specific Domain. Used to be capable of transferring and storing state from one server to another when for instance remoting events are being raised

107. Access

Level2: Encapsulates access right for one single other server in regards to whether or not it can legally raise events, and which events it has requested to raise

108. Gallery

Level3: Tutorial: Contains the Entity Object or Active Type as we call them for our Gallery Plugin. Please notice that the Gallery plugin is more here intended as an example of how to create your own C# plugins than being an actually useful plugin. Especially considering you are literally way more powerful yourself using the Meta Form Builder and such for creating your own types of galleries. Though it serves as a useful example still for how to create your own C# plugins, if the back-end webparts just are not powerful enough for some tasks you need to solve

109. LogItem

Class encapsulating log items in Magix. Magix has automated logging components which will log certain aspects about your system and its behavior. These log items are stored in your database through using this active type

110. MetaForm

Level3: Encapsulates the logic for serializing and storing Meta Forms objects

111. MetaObject

The storage for the 'Meta Application System' in Magix. Every time you create a new object, using a Meta View or something similar, a MetaObject is being created to hold this data. A Meta Object is basically an object 'without structure', where you can, by editing your views, maintain the structure you wish for your data. In such a way you can create your own Meta Applications, using Magix, without being imposed onto an existing data-structure in any ways

112. Action

Contains the serialized content of the Meta Action system in Magix. One Action is a wrapper around an Active Event, as described in the O2 Architecture Document. Basically, an Action can be thought of as a 'serialized version' of an ActiveEvent being raised, which you then can choose to raise whenever you wish, at the trigger of a button for instance. By creating your own actions, you can effectively implement your own type of logic and flow in Magix without having to code. Every Action must have a unique name, and is often either referenced by its Name or its ID property.

113. MetaView

Contains the logic for our views in our 'Meta Application System'. A MetaView is something that defines what the user is supposed to see in regards to MetaObjects. Basically the UI 'representation' of your object graphs and properties. A MetaView can become the 'engine' for a WebPart through the MetaView_Single and MetaView_Multiple plugins for Magix

114. User

Level3: Encapsulates one user in Magix' Publishing system. Inherited to add more, specific to the publishing system, to the class. PS! This class is a fine example of inheriting ActiveTypes for the curious C# developer out there. Basically you've got to override the TableName to point your inherited type to the correct typename, in addition to overriding a "mapped type" in your web.config

115. WebPageTemplate

Serves as a 'recipe' for a WebPage. Every WebPage is built from one of these. Contains the definition of which module to instantiate for instance, but not which parameters to send into it. Also contains the positioning of the WebParts and other common features, such as CSS classes and such. A Template is just that, a 'template' for your WebPages

116. WebPart

Is one instance of a container on a WebPage. Baically one 'module' if you wish. A WebPage is created of several WebParts, each WebPart again is built from a corresponding WebPartTemplate associated with the WebPageTemplate the page itself is built upon. This class encapsulates the WebPart of this relationship, and hence serves as a wrapper for WebPart settings basically on a per page level

117. WebPartTemplate

Serves as a 'recipe' for WebParts. Contains stuff such as positioning of webpart, and name of module to inject. Every WebPageTemplate has a list of these guys, which serves as recipe for how the WebPage's WebParts should be built, and which Plugins to load up

118. OpenIDToken

Encapsulates an OpenID Token. Meaning an OpenID Claim. Magix has support for serving as both an OpenID Relying Party and an OpenID Provider. This class serializes the OpenID Claims for users when using Magix as an OpenID Relying Party, and associating these claims with specific users. This means you can associate an OpenID claim to your user and then use the OpenID Token to log into Magix later

119. WebPage

Represents one web page in our system, or one unique URL if you wish. Contains a list of WebParts which again are areas of the screen within your web page. Every page must have at least one webpart, but can have many more. This class encapsulates the logic of 'Pages' in Magix, which is probably easy to understand within the context of publishing or CMS systems. Anyway, a page in Magix might also be a container for your application, plugins and WebParts. WebParts and Plugins again can be either your own creations through MetaViews or something similar, or actual C# code written plugins for your system. One page is often easy to understand if you can perceive it as 'one URL', while it is still much more powerful than any 'CMS Pages' or 'Publishing Pages' out there. Every page is built upon a 'recipe' which is the WebPageTemplate class. The WebPageTemplate class contains the logic for which plugin type it should use etc, while the WebPage contains the settings for the type of plugins instantiated upon opening it

120. WebPageRoleAccess

Gives access to a page according to a specific role. This is the core class in Magix in regards to authorizing access to specific pages. Access in Magix is associated with a User, his Role and the WebPage the user tries to access. You cannot authorize on a more fine-grained level out than WebPages, out of the box. This class encapsulates the authorization logic in Magix. By default everyone has access to everything. If one Role is explicitly granted access, then all other roles will be denied access, unless they too are granted access. In addition access is 'inherited' from the 'Mother Page'. Meaning a child page will by default have the same access rights as its Mother page, unless the child page itself starts granting and denying access to specific pages

121. Posting

Level2: One Talkback posting. If Parent is null, this is a top-level posting, and the Children collection might have content. If not, it's a child itself of another top level posting, as in a 'reply'

122. UserSettings

Level3: Settings stored on a 'per user level'. Useful for storing simple facts on a per user level

123. UserBase

Level3: A User is a single person using your website, being registered with a username and a password, and hence no longer 'anonymous'. A user is normally recognized through his 'Username', and he has a password which he can use to log into the system, normally. This class encapsulates that logic. PS! This class supports inheriting [though it's not really entirely stable quite yet ...!]

124. Role

Level3: Contains the roles i the system. Every user belongs to a 'role' which gives him rights in regards to some aspect of functionality or something. All autorization, by default in Magix, will go through this Role class, and which specific roles the currently logged in user belongs to

125. Criteria

Level3: Abstract base class for all data storage retrieval criterias. Also contains several handy static constructors for easy creation of data storage retrieval criterias.

126. Transaction

Level3: Implements transactional support for your updates and inserts. Use through the C# using keyword to get automatic rollbacks. Or implement finally yourself in your code. Remember to call 'Commit' before Transaction is lost, since otherwise. Caputt. Default is Rollback

127. ActiveTypeAttribute

Level3: Mark your well known types or entity types [or more correctly said; Active Types ;)] with this attribute to make them serializable. In addition you must inherit from ActiveType with the type of the type you're creating as the generic type argument. Notice that this attribute is for classes, you still need to mark every property that you wish to serialize with the ActiveFieldAttribute.

128. ActiveType-g

Level3: Inherit your well known types or entity types - the types you want to serialize to your database from this class giving the generic argument type as the type you're creating. Notice that you also need to mark your types with the ActiveRecordAttribute attribute in addition to marking all your serializable properties with the ActiveFieldAttribute.

129. ActiveFieldAttribute

Level3: Used to mark entity objects as serializable. If a property is marked with this attribute then it will be possible to serialise that property. Notice that you still need to mark you classes with the ActiveRecordAttribute. Also only properties, and not fields and such can be marked as serializable with this attribute.

130. Adapter

Level4: Abstract base class for all Database Adapters in Magix-Brix. If you wish to build your own data adapter then inherit from this class and implement the abstract methods, add up a reference to the dll and change the data-configuration line in your configuration file and it should work. Class provides some common functions needed for all database adapters like caching, instantiation and such in addition.

131. TransactionalObject

Baseclass for 'internal usage' behind ActiveTypes. Definitely candidate for refactoring. In general do NOT reference any members from here directly, except the ID of course

132. MSTransaction

Level4: Implementation of Transaction class for MS SQL DataAdapter

133. MSSQL

Level4: Microsoft SQL Server Database Adapter, which probably works with 2005 and any later, for Magix-Brix. Contains all MS SQL specific logic needed to use Magix-Brix together with MS SQL. This is the default DataAdapter in use in Magix

134. Helpers

Level3: Static helper class for data-storage Adapter developers. If yo're fiddling around here, you'd better know what you're doing ...!! ;)

135. StdSQLDataAdapter

Level4: Common logic for all Database adapters that relies on standard SQL syntax. Wrappers for creating SQL text for derived data adapters that uses RDBS that relies on standard SQL syntax. Sorry, I have to prioritize what I comment. Needs refactoring though, that's one thing but so does the entire Magix.Data namespace ... :(

136. ActiveModule

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your Modules, such as RaisingEvents etc. Inherit your ActiveModules from this class to simplify their usage

137. ActiveEvents

Level3: Class contains methods for raising events and other helpers, like for instance helpers to load controls and such. Though often you'll not use this directly, but rather use it through helper methods on your ActiveControllers and ActiveModules

138. ActiveModuleAttribute

Level3: Mark your Active Modules with this attribute. If you mark your Modules with this attribute you can load them using the PluginLoader.LoadControl method. This is the main attribute for being able to create ActiveModules

139. ActiveControllerAttribute

Level3: Mark your controllers with this Attribute. Notice that an Active Controller must have a default constructor taking zero parameters. This constructor should also ideally execute FAST since all controllers in your Magix-Brix project will be instantiated once every request.

140. ActiveController

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your controllers, such as Loading Modules, raising events etc. Inherit your controllers from this class if you'd like to add more 'power' to them

141. ActiveEventAttribute

Level3: Mark your methods with this attribute to make them handle Magix.Brix Active Events. The Name property is the second argument to the RaiseEvent, or the "name" of the event being raised. You can mark your methods with multiple instances of this attribute to catch multiple events in the same event handler. However, as a general rule of thumb it's often better to have one method handling one event

142. EventArgs

Level3: EventArgs class that will be passed into your Magix-Brix events - the methods you mark with the ActiveEvent Attribute. The Extra property will contain the "initializationObject" passed into the RaiseEvent.

143. PluginLoader

Level4: Helps load UserControls embedded in resources. Relies on that Magix.Brix.Loader.AssemblyResourceProvider is registered as a Virtual Path Provider in e.g. your Global.asax file. Use the Instance method to access the singleton object, then use the LoadControl to load UserControls embedded as resources. Kind of like the Magix' version of Page.LoadControl. Can be used directly by you, if you really know what you're doing though. In general, I'd say DON'T ...!!

144. AssemblyResourceProvider

Level4: Helper class to make it possible to load controls (and more importantly) UserControls which are embedded as resources in DLLs. Not intended for direct usage, but will be in 'the background' and making sure you can load ActiveModules as resources from your DLLs

145. PublisherPluginAttribute

Level3: I'd be highly surprised if this is not your first entry to Magix in C#. This is the PublisherPlugin attribute, which you can use to create your own plugins for the Publishing system within. Implement this attribute on your ActiveModules and VOILA! They'll surface up as selections in your WebPageTemplate editing operations and be usable as plugins in your system. Probably the easiest way on the planet to create a plugin for any kind of system out there. Especially in combination with the logic behind the ModuleSettingAttribute

146. ModuleSettingAttribute

Level3: Wraps a setting property for a PublisherPlugin

147. PeriodCollection

Level3: A collection class of Period types. Contains algebraic methods for OR, AND, XOR, NOT. Makes algebraic operations on collection of Period objects very easy and intuitive. Can with for instance one line of code OR two collections together to find the logically OR'ed result of these two different collections. Very useful for manipulating dates and such

148. LazyList-g

Level3: Helper class for Lazy Loading of Child ActiveTypes objects. Basically just a list generic type, that'll not load objects before needed. Useful for using as properties for list of child objects in your ActiveTypes

149. Node

Level3: Helper class to pass around data in a "JSON kind of way" without having to convert to JSON strings. Create a new instance, and just start appending items to it like this;

```
Node n = new Node();  
n["Customer"]["Name"] = "John Doe";  
n["Customer"]["Adr"] = "NY";
```

This is at the core of Magix, being the 'protocol' we're using to pass data around within the system. If you don't understand this class, you're in trouble! Make sure you understand, at least roughly, what this class does if you'd like to code C# for Magix

150. SingleContainer

Level2: Contains the logic for the main Viewport in Magix. A viewport can be seen as your 'design' and contains all the different logic for being able to load and unload modules and such

151. AspectModal

Aspect useful for making 'modal' controls. A Modal control will obscur the contents behind it by creating a div that is semi-transparent (most often) and fill the entire viewport (browser area). This effect makes it impossible to click controls behind it on the page.

152. AspectAjaxWait

Ajax Wait Aspect. Use this one whenever you've got an Ajax Control that you know will spend a lot of time on the server-side executing its logic. You can either have the client-side logic create its own DOM element to serve as a 'blackout' when it's supposed to display the Ajax Wait DOM element, or you can attach an existing in-visible DOM element to it by using the Element property.

153. AspectBase

Base class for all Aspects in Magix UX. Implements common functionality.

154. AspectDraggable

Aspect for creating controls that can be moved around on the screen using your mouse. It is also possible to handle the Dragged event and track when the dropping of the control occurs.

155. Uploader

Level4: Widget encapsulating an HTML5 drag and drop File Uploader control, with progess bar and support for multiple files

156. SlidingMenu

This widget is a 'menu looka-like kinda control'. Its purpose is to be an alternative way of displaying hierarchical choices for your users. But instead of 'popping up' like menu items does in a conventional menu, they will 'slide in' from the side, making a much more efficient way of handling your space on your web pages. For most navigational purposes, this widget is a far better bet than the conventional Menu widget.

157. SelectList

A DropDown list type of control, although it can also be set into a non-drop down mode. Basically multiple choices type of widget. To some extent, it overlaps the logical functionality of the RadioButton widget, although the SelectList is more useful for cases where you have a massive number of choices, like for instance choose one out of 300 different languages, while the RadioButton is more useful for cases where you have fewer choices, such as choose 'coffee, tea or water'. Add up your choices by adding up ListItems inside of your SelectList.

158. SlidingMenuItem

A single sliding menu item. One instance. A SlidingMenu are basically composed out of a whole hierarchy of these types of widgets.

159. SlidingMenuLevel

Child control of a SlidingMenuItem. Will contain the SlidingMenuItems of a SlidingMenuItem. If you have items which have children, then you must create an item of type SlidingMenuItem and add those children inside of that object. This is true both for .ASPX markup and SlidingMenu hierarchies created in code.

160. MultiPanelView

Instances of items within a MultiPanel. These are the items a MultiPanel is made of. Most scenarios you would use this control within would have only one of the MultiPanelViews visible at the same time.

161. MultiPanel

A MultiPanel is a collection of MultiPanelViews, often used to create TabControls or similar constructs. Often combined with TabStrip and TabButton. This control can be configured extensively to mimick a lot of different scenarios. Hence we don't call it TabControl, although that happens to be one of the sub-scenarios you can create with it.

162. Panel

A container widget for displaying other widgets inside of it. Will render as a div by default, but the specific tag this widget will render, can easily be overridden by changing the Tag property. You can choose to render your panels as paragraphs (p...) for instance. If you only need to display text on your page, and you need to use WebControls for this, you should use the Label control and not the Panel control.

163. RatingControl

Encapsulates a Rating Control, for allowing the end user to choose number of stars, or 1-x value of something

164. RadioButton

A two-state widget type that can be grouped together with other widgets to mimick multiple selections. Like for instance; "Would you like to have coffee, tea or water" is a perfect example of where you would want to use RadioButtons. If you want to group specific radio buttons together, you must give them the same GroupName property. If you do, then only one of these RadioButtons can at any time be 'selected'.

165. TreeItem

One single item within a TreeView widget. A TreeView is basically nothing but a collection of these types of controls. If you override its CSS class, you can give your TreeItems specific icons and such, in which case you need to change the CSS background-image for those specific MenuItems, which really is out of scope of this article.

166. Timer

Timer widget that will periodically call the server every n'th millisecond. You can set its period through the

167. TreeView

This widget makes it possible for you to create selector kind of widgets that mimicks the representation of a tree kind-of hierarchical structure for displaying things. Useful for displaying for instance folder structures and such.

168. Window

Window control. Basically an "advanced panel" with support for moving. It also features borders which can be skinned. The equivalent of a normal desktop window.

169. TabButton

One instance of an item within a TabStrip. Mostly shown as buttons inside of the TabStrip.

170. SubMenu

Child control of a MenuItem. Will contain the SubMenuItems of a MenuItem. If you have MenuItems which have children, then you must create an item of type SubMenu and add those children inside of that object. This is true both for .ASPX markup and Menu hierarchies created in code.

171. TabStrip

A strip of TabButtons, used often to trigger changing active view in combination with the MultiPanel control. This control is a collection of TabButtons.

172. TextBox

A single-line type of 'give me some text input' type of widget. This widget is a wrapper around the input type="text" type of widget. If you need multiple lines of input, you should rather use the TextArea widget. However this widget is useful for cases when you need the user to give you one line of text input. See also the RichEdit widget if you need rich formatting of your text. This widget can also be set to 'password mode', which means whatever is typed into the widget will not be visible on the screen. Please notice though that by default, this will be transferred to the server in an unsecure manner, so this is only a mechanism to make sure that other people cannot read over your shoulder to see what you're 'secretly' trying to type into your TextBox. Use SSL or other types of security to actually implement safe transmitting of your passwords and similar 'secret text strings'.

173. TextArea

A multiple line type of 'give me some text input' type of widget. It wraps the textarea HTML element. If you only need single lines of input, you should probably rather use the TextBox widget. However this widget is useful for cases when you need multiple lines of text input. See also the RichEdit widget if you need rich formatting of your text.

174. MenuItem

Items within a Menu. A MenuItem can have a Text property in addition to a SubMenu, which if defined will popup as a child of the MenuItem. The MenuItem class is basically the individual items of your menu.

175. CheckBox

A CheckBox is a 'two state button' which you can turn 'on' and 'off'. Useful for boolean UI situations where use must choose between two options, for instance 'yes' or 'no' situations.

176. DynamicPanel

Control for making it easier to dynamically instantiate new controls and add them into your page. Completely abstracts away the entire hassle of storing whether or not a control has been previously loaded into the page or not. Make sure you handle the Reload event, and dependent upon the key given will load the exact same control and add it up into the DynamicPanel, and you should experience a very smooth experience in regards to dynamically loading controls into your page. Call the method LoadControl or AppendControl with a unique ID defining which control you wish to load, for instance the name of a UserControl file on disc, and until you explicitly clear your DynamicControl, the same control will be automatically loaded every time. If you do not create a Reload Event Handler for your widget, and you call LoadControl, then the DynamicControl will assume that what you're passing in is a fully qualified path to a UserControl, and attempt to load it as such. You can use the Extra parameter to add extra initialization parameters into the control upon its first load.

177. ColorPicker

ColorPicker widget for allowing the End User to select a Color from a Color Wheel kind of control. Use the ColorWheelImage to override which colors the user is allowed to pick from, if you'd like to restrict him to some sort of palette

178. AccordionView

Panels contained inside the Accordion control. Basically an Accordion is only a collection of these types of controls.

179. Accordion

Accordions are panels, or AccordionViews grouped together where you can expand and collapse individual AccordionViews within a group. Normally only one AccordionView can be visible at the same time, but this is configurable.

180. Button

A clickable button. The equivalent of input type="button". Use when you need a clickable thing to resemble a button. See also the LinkButton for an alternative. Also remember that any Widget in Magix UX can be made clickable, so you can also use a Label as your 'clickable thingie' if you wish. Even though anything can be made clickable in Magix UX, it is often an advantage to use buttons or link buttons since these elements will mostly be recognized by screen readers and such, and it is hence more 'polite' to use these specially designed types of 'clickable objects' such as the Button.

181. HiddenField

Hidden field widget. Useful for having state you wish to pass on to the client but don't want it to be visible for the end user. Notice that this is not a safe place to put things that the user is not supposed to see, like passwords and such. Do NOT trust the value of this element to not be tampered with. Alternatives for using this widget is ViewState, cookies and the Session object. ViewState and cookies are neither 'safe' against tampering.

182. LinkButton

This widget is another type of 'button widget', though this will be rendered using anchor HTML element (anchor element...) Even though everything can be made 'clickable' in Magix UX, it is definitely semantically much more 'correct' to constraint yourself to the ones that are expected to be 'clickable', such as this widget (LinkButton), Button, ImageButton etc. Among other things screen-readers and such will recognize these types of elements as 'clickable' and present the end user with the option of clicking these types of widgets.

183. Label

A 'text widget'. The basic purpose of this widget is purely to display text and nothing else, though through the CssClass property and the Style property you can easily manipulate this to do mostly anything you wish. If a more 'complex widget' is needed, for instance to host other widgets, then the Panel widget is more appropriate to use than the Label. Unless the Tag property is changed, this widget will render as a span...

184. Menu

Menu control. A Menu is basically a collection of MenuItem's. Mimicks a Menu the way you're used to seeing them on desktop systems, or at other web applications. The Magix UX menu have support for any number of child menus, and are very customizable in regards to how it looks and behaves.

185. Image

Image Ajax Widget. Useful for showing images that needs Ajax functionality somehow. Notice that most times it's more efficient to display other types of widgets, such as the Panel or a Label and set it to display an image through using something such as background-image through CSS or something similar.

186. HyperLink

A wrapper around a hyper link or anchor HTML element (anchor HTML element ...) Sometimes you will need to create links that might change or needs changes after initially created. For such scenarios, this widget is highly useful.

187. ImageButton

Technically this widget is completely redundant towards the Image Ajax Widget. But it is here for cases where you want to be semantically highly correct and need something that gives clues in regards to that it is 'clickable'. If you have an Image which is clickable, then semantically this widget is more correct to use.

188. CompositeControl

Abstract helper widget for widget developers. Used internally in e.g. Window and Accordion. Useful for those cases where you have a widget which is 'composed' out of both a norml 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself. The Window is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Caption and a Close button. That's why the Window is a 'CompositeControl'.

189. BaseWebControlListFormElement

Abstract base class for widgets which are HTML FORM or multi-choice type of elements

190. ViewCollectionControl-g

Abstract base class for widgets which are supposed to contain only one type of child widgets. Contains a lot of usable shorthands like for instance Enumerable support, and an index operator overload.

191. CompositeViewCollectionControl-g

Abstract helper widget for widget developers. Used internally in e.g. TreeItem. Useful for those cases where you have a widget which is 'composed' out of both a normal 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself, and you in addition to this also have a specific type of widgets that are supposed to be its child control collections, like for instance the TreeItem are only supposed to have TreeItems within itself. The TreeItem of the TreeView is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Text property and many other child controls. In addition it is supposed to only have child controls of type TreeItem. That's why the TreeItem is a 'CompositeViewCollectionControl'.

192. BaseWebControlFormElementText

Abstract base class for WebControls which are BaseWebControlFormElement type of controls, but also have a Text property. Text property is overridable, its default implementation sets the 'Value' property on the client-side, which may or may not be suitable for your needs.

193. BaseWebControl

MUX WebControl equivalent. Contains a couple of additions to the MuxBaseControl. Among others a style attribute and a CssClass. Also contains most of all the DOM event handlers you can possibly handle in Magix UX. In addition to most of the client-side effects you can possibly run on your MUX controls. Class is abstract, and hence cannot be instantiated directly but is meant for inheriting from.

194. BaseControl

Abstract base class for mostly all widgets in Magix UX. This is where the Ajax 'core engine' to a large extent exists within Magix UX. Contains several interesting features, such as for instance the Info property, which can take any string information and serialize back and forth between server-requests. If you need to create your own Ajax Control, you should either directly, or most probably indirectly, inherit from this MuxBaseControl. If you inherit, directly or indirectly, from this class you need to override several methods from this class. The most notable is RenderMuxControl. If you have a 'visual widget', you will mostly inherit from MuxBaseWebControl instead of this class. If your control is 'non-visual', such as the MUX Timer, you will mostly inherit from this class.

195. BaseWebControlFormElementInputText

Abstract base class for widgets being BaseWebControlFormElementText type of controls, but also uses the Text portions as an input value which the user can change himself through interacting with the widget.

196. BaseWebControlFormElement

Abstract base class for widgets which are HTML FORM type of elements. Abstracts away things such as blur and focus event handling, enabled, keyboard shortcuts and other internals things. Inherit from this one if you intend to create an Ajax Widget which wraps an HTML FORM element.

1. Welcome to Magix

Magix is a platform for Creating Web Applications, either by using C#, VB.NET or other .Net languages, or through using one of the No-Code-Required Application Creators, which you can use to create Web Applications without coding.

I am Heka, the Anthropomorphized Soul of Magix Illuminate I. By using me as your guide through Magix, you can become friend with Magix faster.

In fact, your first Tip, is to use the Arrow Keys in the Top/Right corner of this window, to navigate forward and fast read the next upcoming 5-10 tips. They are important for getting started with Magix ...

If you need to re-read a previous tip, you can click the previous button ...

Magix is ...

Magix Illuminate has many features, abd trying to list them all, would demand many books. Though already before you need to create anything yourself, Magix Illuminate is already extremely useful. You can for instance use Magix ...

- As your own personal DropBox server - By using the File Manager
- To Synchronize your files between work and home
- File Sharing with Friends and Colleagues
- Cloud-Backup of your Important Files, such as Images
- MP3 player, which will let you play your music from every device you wish
- Gallery for friends and family
- Vanity QR Code Generator
- CMS or Publishing System
- Micro Website CMS
- Excel CSV File Manipulator

All these things comes in addition to the endless possibilities you have for creating your own Applications, either without code, or by using any .Net CLR language.

Creating Stuff

Magix consists of two important pre-installed modules; The 'Publishing' System and the 'MetaType' System.

Publishing is where you go to create WebSites or to publish text, images and videos. MetaTypes is where you go to create Applications.

We recommend you to start with learning the Publishing concepts first. Then later, only when a firm grasp of Pages and Templates are understood, moving onto Applications. Although, if you have had much experience with Publishing Systems before, you might want to rush this part. The basic understanding of Pages should be easily acquired for a Savvy website owner relatively fast.

Basics ...

Before we can do anything, we need to learn the Basics ...

Most of Magix is made up of 'Basic Components', which are tied together to create a whole. For instance, a button will mostly look the same everywhere. By default a button will be Gray and use Blue, Big Fonts. A good example of a Button is the Top/Right corner of this tooltip, which has two Buttons. One Paging forward, and another paging Backwards in the Hierarchy of Tips and Tricks ...

A 'Grid' is when you see a list of items. A good example of a Grid would be Actions or Views. Grids are useful for displaying lists of information.

There are many types of basic components like these in Magix. Over the next couple of pages, we'll be walking through some of them which you'll need to understand to be able to get the most out of Magix ...

Grids

Most Grids in Magix have many features. These features includes paging, in place editing, filtering, etc. Open up Actions by clicking the icon at 2 O'Clock on your Dashboard.

To Filter according to a column, all you've got to do is to click the header of your Grid, choose which type of filter you want to apply, type in its value, and click OK. Try to filter your actions according to 'debug' by clicking the name header column. This should give you two actions, turn on and turn off debugging. Though some screens, such as the action screen, also have a 'fast filter' textbox. Make sure you remove your filter before you leave this screen again, by clicking the name column and remove the text and click OK.

The arrow buttons, normally at the bottom of your grids makes it possible to traverse forward and backwards in your list of items. The double arrows takes you 'all the way' in its direction

Open up Objects/Actions and play around with that grid by filtering, creating a couple of new items and so on.

Paging normally turns on automagically according to when it's needed. To see the paging buttons, you'll normally need to have more than 10 items in your grid. If you don't see them in your Action screen, make sure all Filters and filter queries are turned OFF. To see 'all the way paging', you'll normally need more than 20 items ...

Make sure you also click the 'Edit' column. Sometimes this column will say 'Edit' while sometimes it'll show a number. However, clicking the Edit Column, will always somehow bring you to a View where that object can be edited in 'full version' ...

Some things should be very similar for mostly all grids like this in Magix. For instance ...

Clicking the '+' button will almost always create a new object of that type ...

If the Text of a Grid Cell is Green, this means that you can edit the value directly by clicking the Blue Text, which will exchange it with a 'textbox', from where you can edit its value, and save the updated value by either clicking Carriage Return or somehow moving focus away from the textbox/textarea itself. For textarea (multiple lines textboxes) hitting the TAB key to update is usually the most efficient ...

Publishing ...

A Website consists of Pages. Every Page is the equivalent of one 'URL'. Although URLs doesn't really exist in Magix, it helps to think of a page as such. Beside, creating a URL based Navigation Plugin would be piece of cake anyway, due to the Architectural Principles Magix is built on ...

Anyway ...

You create your pages according to 'Templates', which can be seen as 'Recipes' for your pages. You have to have at least One Template in your system before you can start creating Pages. Every Page is based upon a Template, and no page can exist without its Template ...

Click Templates at 3 O'Clock on your Dashboard now.

BTW, Click the Edit links to view any specific templates ...

Every Template contains a list of WebPart Templates. These WebPart Templates can be positioned exactly as you wish on your page. When editing a Template, use the Arrow Buttons to position your WebParts ...

If you click the Blue Name of the WebPart, you can further edit more properties of your WebParts. The most important property here is the Module property, which basically is the type of Publishing Plugin Module this specific WebPart is based upon.

Some of the more important types of Modules are

- MetaView_Form - A Publishing Plugin-Host for your Meta Forms. What allows you to inject Meta Forms into your Pages
- Content - A Rich Text content plugin
- Header - Basically just encapsulates an H1 element
- SliderMenu - The Sliding Menu, which can be seen by default in both the back-web and the front-web of Magix
- TopMenu - And Alternative and more conventional Menu for creating Menu Hierarchies. 100% compatible with the Sliding Menu's Data Foundation

But exactly which modules you've got in your installation, may vary according to what modules you've installed, and how you've modified it yourself after you installed Magix Illuminate.

Another important property is the WebPart CSS Class, which also can be manipulated using the CSS Template SelectList. The Last property should only be set for WebParts which are supposed to expand all the way to the right of your Viewport. While the Overflow is for dynamic heights, where you don't know the final height of your module's content

When positioning your WebPart Templates, realize that you're not really 'positioning' them, but rather you are changing their width, height and margins. In the beginning this might feel a little bit cumbersome, though after some time you'll hopefully appreciate this 'floating layout' and become used to it ...

Realize also that especially the bottom and right margins might create funny looking WebParts since they're not really visible while editing. If you're having weird results, make sure your right and bottom margins are 0 by double clicking them, which should set them back to zero ...

Double clicking any of the arrows will either maximize or minimize their associated property ...

As previously said, realize that you're not really positioning your WebParts when you click the Arrow Keys. What you're doing, is that you are setting the Margins and the Width and Height of your WebPart. Though both the left margins and the top margins can have negative values, which basically is the facilitator to become virtually completely 100% enabled to set your exact positioning, without residing to absolute positioning.

A little bit more cumbersome, but definitely worth it.

The Grid Layout System

When you move things around, either by using CSS or some other helper methods, such as the Arrow Buttons in the Template Editor, then you are basically conforming to a Grid system which Magix has inherited from Bluegrid.Css, which it uses at its core.

This system basically is 950 pixels wide, which happens to be 'the perfect' resolution for all monitors and devices, including phones and tablets.

These 950 pixels is divided into 24 columns, where every column is 30 pixels wide, and have a margin of 10 pixels. Meaning if you're covering one column, your WebPart is 30 pixel wide. If it's covering 2 columns it's 70 pixels, 3 columns 110 pixels, and so on. Pluss 40, all the way to 24 and 950 pixels.

Every row is 18 pixels tall. Some things such as Buttons, by default, will cover two rows, meaning 18x2, which becomes 36 pixels.

When you move your WebParts, what you are really doing, is that you're adding and removing left and top margins of 40 and 18 pixels for every 'movement' you create. This allows for the perfect reading layout that Magix Illuminate comes with out of the box. Also for your own apps and pages.

If you wish to override this, you can do so either using CSS on the WebPart and WebPage level, or you can change the absolute positioning by allowing Absolute Positioning in the Meta Form designer for instance. Though this is generally not 'encouraged', since it destroys your future Agility.

A WebPage Template must contain at least one WebPart Template. A WebPart Template is also a 'Type Definition' for your WebParts. WebPart Templates have names such as 'Content' and 'Header', which are publishing modules for showing large letters and rich text fragments. Every WebPart Template is based upon one plugin type.

Meaning if you have one page, based upon a WebPage Template, with 5 WebPart Templates, you'll have a WebPage with 5 WebParts where each WebPart can be different 'Applications'. This might be any combination of Applications, such as Text Fragments, Headers, CRUD Forms and such ...

However, we'll stick to 'Publishing' as we promised in the beginning, and focus on the Publishing Modules ...

If you edit the default Template created by the system for you, you can see how it has Menu, Header and Content as 'Module Names'. You can also create as many new Templates as you wish.

If you open your Dashboard now, and click 'Pages' at 1 O'Clock, you'll come to your WebPages Editing Screen. Click the top one and see how it's built upon the Template called 'M+H+C', which means Menu + Header + Content btw.

Every time you create a new WebPage, which is a single adressable URL in your system, you'll have to decide which Template to base your Page upon. Which again defines your design, type of applications this specific WebPage can show, etc.

The Menu is similar to the Sliding Menu to the left which you're using yourself, while the Header will show an H1 HTML element [Header Element] and the Content module will show Rich Editable Content.

If you have more types of Modules in your installation of Magix, these might also show up as selections in the DropDownBoxes visible while editing your Templates ...

Now go to Templates and change the Name of Template 'M+H+C'. Change it to 'Testing'. This can be done by clicking directly on the text where it says 'M+H+C'. Then open up 'Pages ...' and click your root page.

In your Pages view now, can you see how the name of the module in the DropDown box, roughly at the middle of the screen has changed now to 'Testing'. This is because that DropDown box is being used to select a Template for your WebPage.

Now try to change the Name of your Templates by clicking e.g. Header while editing your Template, and type in 'Header2'. Now edit your Page and see how this change reflects from the Template and to the Page.

So to conclude, one WebPage is based upon a Template, which again contains several WebParts, where every WebPart is based upon a Module Type. The Page defines your content, and the Template your design, and 'type' of WebPage.

Important: If you change the number of WebParts in your Template, or you change the type of Module of your WebPart Template, then all pages built upon that Template will have to be resaved, and you'll probably loose data.

It is therefor important that you create your templates first, and then don't edit these two properties while they're already in 'Production' ...

Try to Create several different Templates, and have slightly different values for their width, height, margins and such.

Be certain of that you've added different widths of your Menu Containers and different Left Margins

Make sure they've got the same type of modules in the same container

Then use these different Templates for different pages which you create in your Pages hierarchy

If you now access the root of your website, and try to browse around by clicking different buttons, you can see how the WebPart Containers are 'jumping around' on the screen ...

Initialization of WebParts

WebParts initializes differently depending upon where you're coming from, and from which Template type you're coming from dependent upon to which Template type you're entering. And also according to which type of WebParts, or Modules they are ...

For instance the sliding menu will not reload as long as the container it is within on one page template, is the same container it is within on the next page template ...

Too confusing ...?

Just remember; always have a Menu positioned in the same container [first one for instance?] on all of your templates, unless you really know what you're doing ...

Play around with the system by creating some new Templates, copying them, change their weparts type between Header, Content and SliderMenu. Then when you come back, we'll start diving into Applications ...

Applications ...

Most applications will be WebParts. This means that you can inject them into any WebPart onto any WebPage you wish.

Let's create an Application ... :)

First make sure you have one Template in use in one of your pages which has one WebPart Template with the Module Type of 'MetaView_Single' ...

Then click on 'MetaTypes/Meta Views' ...

Create a new View called 'CollectEmails' ...

Add three properties to your form, name them

- Name
- Email
- Subscribe

Change their description to something meaningful ...

Make sure you change the 'Type Name' of your object to 'EmailSubscription'

Attach two Actions to your 'Save' property.

- `Magix.DynamicEvent.SaveActiveForm`
- `Magix.DynamicEvent.EmptyAndClearActiveForm`

The first Action will save your form, while the second one will empty it.

Now try to View your form in preview mode, and test it out by typing in your email and name, and clicking Submit to save your Object ...

BTW, Obviously it's crucial that the 'Save' action runs before the 'Empty' action, in case you wondered ... ;)

Type Names

If you take a look at your Meta Objects now you will see a new object with the Type Name of 'EmailSubscription'. The 'Type Name' property from your MetaView decides the Type Name of your Objects ...

These 'Type Names' are important to distinguish from different types of Objects.

One Object might be of type 'Customer', while another object might be of type 'Email', and so on. What names you give your Types is crucial! Name clashes here might create very hard to track down bugs and such ...

Take some care when naming your Types!

It's probably a good practice to somehow make sure they've got unique names, also across your organization if you want to use plugins made by others.

We encourage people to use type names such as; "CompanyName.Department.Customer", and never 'Customer' directly. In fact, your homework for this lesson is to go and rename your 'Customer' TypeName, and rename the Type Name to; 'CompanyName.Department.Customer'. Where Company Name and Department are your company name and your department ...

Expressions

There are several places where you will need to understand Expressions. Expression starts out with '{' and ends with '}'.

The purpose of an Expression is to de-reference something within your existing Node structure. Every Module and Action has a Node associated with it, which can either be statically, or dynamically built. This Node can be seen as the action/module data source.

If you run TurnOnDebugging in your Actions, you can see how this Node structure is being passed back and forth by copying the Active Events with all their Parameters, serialized with the correct state, accordingly to how they looked like when they were being executed.

This statement; 'root[Customer][Name].Value' means you will de-references the Value of whatever is inside the Root node of either your action or your module, within its Name node, which should be within the Customer node on root level. If this value, or these nodes doesn't exist somehow, a null value will be returned.

If you start your expressions with 'root', it will traverse all the way upwards until it finds the top outer most node. Otherwise it will always expect to find its node relatively according to where it is while traversing.

You can go up one level by using [../]. What you end your expressions with defines what will be returned. There are three basic ways you can end your expressions, '.Value' to return the Value of the Node attempted to reach, '.Name' for its name, and nothing ']' to return a list of nodes. Not all operations can legally accept all different versions of expressions, just because they accept expressions. This is however individually for Active Event.

Magix Turing Executor

The Magix Turing Executor is probably among some of the more complex, but also powerful concepts in Magix.

It is basically the equivalent of programming, except instead of having to go by 'code', you create your execution tree directly as a tree structure, and you can intermingle Data and Code as you wish, and have code that creates code and such as much as you want to.

Which btw, for the record, also opens up a whole range of security problems if you allow for code to be executed this way from external sources. Be careful!

As everything else in Magix, also the Turing Executor is started with an Active Event. Search through your Actions for execute to find a template action, which you can copy to create your own Turing Executable actions, which can contain Business Logic and all types of branching logic.

The Magix Turing Executor has these keywords

- if - Checks a 1 or 3 compound expression and executes the given tree node as if it was an execute block if the expression evaluates to true. 'if' has these legal operators '==', '!=', '>', '<', '>=', '<='. All comparisons are currently being done by string comparison, meaning alphabetically, meaning you cannot compare numbers unless they're somehow formatted to match up in regards to digits alphabetically. E.g. filled.
- else if - Same as above, but will only be evaluated if the previous 'if' statement did not evaluate to true. 'else if' obeys by the same rules as the 'if' statements otherwise. An 'else if' statement must follow either an 'if' statement or another 'else if' statement, otherwise an execution error will occur.
- else - Same as above, but has no expression, and will be evaluated if none of the previous 'if' or 'else if' evaluated to true. Default execution path.
- execute - Creates a new block of execution. Use Value as comment if you wish.
- set - Expects an expression as a left hand, a '=' operator, and either an expression or a static value as its right hand expression.
- raise - Will raise the Active Event defined in its Value, with the given context node as its paramaters, unless the Active Event contains an expression at its end, inside of parantheses(), in which case it will pass on the Node expected to be found at its expression.

Drag and Drop Files

There are many places in Magix where you can Drag and Drop files. One such example would be the File Manager. If you open up the File Manager, you can find the folder you wish to upload your file(s) to, then find the files on your Desktop System, and actually drag and drop the files you wish to upload to this folder on your server. Magix will take care of the rest.

If you want to use these features in your own apps, they are possible for you to use by for instance adding a WebPart to your Template and set its type to Uploader.

OpenID

Did you know that you can use Magix as both a Relying Party and OpenID Provider?

[Read more about Open ID here ...]

If you need an OpenID token to log into some website somewhere, then you can append ?openID=admin after the root URL to your website, if the admin user is the User you'd like to log in with.

[Full example; yourdomain.com/?openID=admin]

This will redirect from the website you're trying to log into, and back to your website, which in turn will ask you for admin's password.

Once successfully logged into your own Magix website, your website will redirect back to the website you're trying to log into, and tell it that it 'has proof of that you are who you claimed you were'.

You can also associate other OpenID tokens, such as your Yahoo account or Blogger account, with your Magix User internally. This will allow you to log into Magix with that OpenID Account.

If your blogger account is 'magix' for instance, then your OpenID 'username' [claim] would become 'magix.blogspot.com'

Talk to CEO

If you make something cool with Magix, that you want to share, then we'd love to get to know about it. It can be an instructional YouTube video about how to Get Started, Tips and Tricks etc. It can be a Tutorial Blog you've written about how to install Magix on your server. It can be a book you have written about the O2 Architecture. It can be an Action, or a collection of Actions, which you think is awesome. Anything really!

As long as it has Value for our Community somehow, we'd love to know about it, so that we could help you promote it, and more importantly; Helping our Community out ... :)

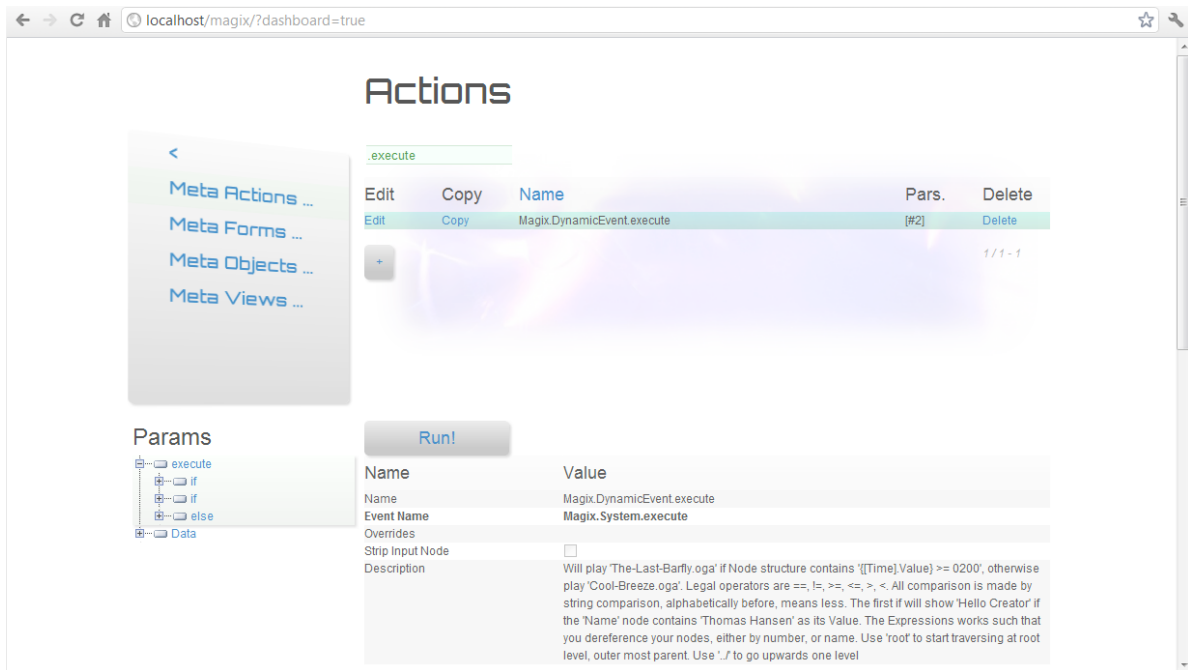
Our CEO's Name and Email address is; Lissa Millspaugh - lissa.millspaugh@winergyinc.com

Our CTO's Name and Email address is; Thomas Hansen - thomas.hansen@winergyinc.com

If your stuff is of 'geeky nature', it's probably best to send it to our CTO, or at least CC him in ... ;)

2. Meta Actions

Actions is one of the more important concepts in Magix. Actions wraps Active Events, which again wraps verbs, and hence becomes all your Business Logic. Combined with the Meta Form Designer, Actions are the by far most important concepts you'll need to understand before you can achieve productivity in Magix.



While logged in as an Administrator in your Dashboard, click either 'Actions' or use the menu to open up your Action list.

Meta Actions, or 'Actions' for short, are dynamically created actions within the system, either created by the system itself during installation, or Actions created by other software packages, or by users, such as you.

System actions cannot be changed or modified through the Action screen. To change a system action, copy it and change the copy and consume the copy, or use the Data Explorer, which will allow you to edit your System Actions - NOT RECOMMENDED!

Actions can be invoked by some sort of event, for instance a user clicking a button, or something similar, in for instance a Meta Form or a Meta View. Basically, if it can be described as a verb, it's an Action.

Actions are wrappers around Active Events, which again are sinks within your code, which you can reach from the outer most environment as you wish, depending upon what you wish to allow for, as an admin of your server. The Action System is the soul of Magix, since it's the wrapper around the most important concept in the O2 Process; Active Events.

Actions can override Active Events, and polymorphistically transfer execution to another server of your choice, and such create perfect scaling out capabilities, 100% transparently. To use this feature search for the 'transmit' action in your Actions list. The Transmit Action shows how you can transfer polymorphistically an action from one server to another using HTTP POST and serializing the Active Event Name and Node structure down to JSON and HTTP POST parameters such that the event is 100% transparently transferred to another server of your choice, and invoked at your endpoint, if permissions are granted, and then returning the result back to the invoker of the Action.

By combining Transmit with Execute actions, you can create logical entities such as 'Customer Server(s)' or 'Logging Server(s)' etc, simply by polymorphistically raise your Active Events on another end point of your choice.

Actions can override one Active Event with another and such exchange one piece of logic with another piece of logic. To use this feature make sure the 'Override' property of your Action points to the Action you wish to override, then run the ReInitializeOverridden system Action. No changes are being done by Overrides before ReInitialize is executed. This is such that you can make changes in your system, then later when you're done, you reinitialize your actions overrides, and your system updates with the new logic.

Actions can execute dynamically created Executions trees to create Turing Complete systems. Search for 'execute' to see an example of this. Execute allows you to create a Node structure with control mechanisms such as 'if', 'else if', 'raise', 'throw', 'foreach' etc. The intent of the Magix Turing Executor is to allow for the system to become 100% Turing Complete without needing to resort to code, while still keeping enough structure to be capable of completely abstracting it away through visual WYSIWYG type of constructs. This combined with the serialization support of Magix for parameters and function

invocations and results and such, combined with Magix' extreme meta support, makes it relatively easy to create really interesting stuff such as Virtual Reality construction of executable code, which is 'recorded' as the individual opens and closes doors which represents your legal execution logic flow, etc, etc, etc.

All Actions you create will have their Description become an integral part of the documentation.

To summarize, such that you can more easily remember and understand Actions conceptually, Actions are partially serialized function invocations, which can be invoked again and again and again. Parts of their Node structure will automatically be transferred by the invoker, and carry over stuff such as the DataSource of the Active Module etc, while other parts are statically defined through the static Node parameters of your Actions. Any amount of runtime polymorphism can later at any point be added, either by an automatic process, or by the administrator of the server during configuration.

This extremely flexible late binding polymorphism mechanisms of Magix is really what makes Magix Magic.

Extremely easy to understand conceptually, after some initial 'aha' time. And insanely flexible, after the first wave of 'ahas' have passed. Realize though that yours truly is still doing 'ahas' in regards to Actions. Actions will, and should, largely replace your functions and methods. Actions basically replaces every single OOP mechanism you have used in conventional OO languages. You can still use OOP constructs as you wish in Magix, but Actions and Active Events will most of the time become your preferred weapon of choice. Simply because of their superiority to old-style functions and methods.

Yes, they're also extremely fast.

The different properties of your actions are

- **Name** - Serves as a unique reference for later being able to invoke the Action. The Action's function name, if you wish
- **Event Name** - The Active Event the Action will raise upon invocation
- **Overrides** - If not empty, will become the polymorphic overridden implementation for the given Active Event, with the same name as the Overrides property
- **Strip Input Node** - If true, will strip the incoming input node from the place which raised the Action

If you're not editing a System Action, then you can use the Search button to search for Active Events which you want to wrap your Actions around.

Creating and Invoking Active Events from C#

When you create an Active Event, you associate it with a name, which is a string literal, which later becomes its 'address'.

```
// This code should either be in an Active
// Controller somewhere, or in an Active Module
// If the code is in an Active Controller then
// it will handle the Active Event every single
// time it is invoked. If it is in an Active
// Module it will only be handled in that Active
// Module if the module is loaded, unless you make
// the method which handles your Active Event
// static
```

```
[ActiveEvent(Name = "YourNamespace.PatPitty")]
protected void YourNamespace_PatPitty(object sender, ActiveEventArgs e)
{
    /* do stuff here. e.Params will contain
    * all the static Nodes you've got in
    * your Action in your visual interface,
    * in addition to that if Strip Input
    * Node is false, the entire
    * Node structure from the caller, normally
    * the DataSource or previous Action
    * will be passed on into here too,
    * with the static parameters overwriting the
    * dynamic ones if needed.
    */
    PetAKitten(e.Params["Kitten"].Get<string>());
    e.Params["Result"].Value = "Moaw!";
}

protected void PetAKitten(string kittenName)
{
    /* Kitty, kitty, kitty ... :) */
}
```

[... some other file somewhere, which doesn't need to know anything about the component implementing the Active Event handler ...]

```
// From some Controller somewhere, inheriting from ActiveControll base class ...
// Or any other place referencing Magix.Core, if you'd like to type out the
// ActiveEvents.Instance.RaiseActiveEvent instead of the shorthands ...
protected void PittyPatty()
{
    Node n = new Node();
    n["Kitten"].Value = "My Kitten";
    RaiseEvent(
        "YourNamespace.PatPitty",
        n);
    ShowMessage(e.Params["Result"].Get<string>());
}
```

The above code is an example of how to create an Active Event, in C#, which again can be serialized to an Action which wraps it.

Simply by realizing that Active Event address units are nothing but strings, you realize you can pass references to methods around, almost like you'd do with a dynamic programming language with lambda support. Except with Magix, you don't need to pass around your Lambda objects, but you can cross-interfere and override at any level, without needing to do changes to code. The entirety of the polymorphism mechanisms are completely runtime configurable, basically ...

By decoupling our polymorphism mechanisms away from our types and objects this way, we achieve exponential productivity growth after some initial learning. Because everything becomes reusable. Just for God's sake, do NOT pass complex types around. If it's not in the .Net System namespace, it probably doesn't belong in the Node structure!

It is considered smart to namespace your Active Event such that they're unique for your company/department/project etc - To avoid namespace clashing.

SaveActiveForm

Magix.DynamicEvent.SaveActiveForm

Magix.MetaView.CreateSingleViewMetaObject

Will save the currently active Single-View Form. Will determine which form raised the event originally, and explicitly save the field values from that Form into a new Meta Object with the TypeName from the View ...

ReInitializeOverriddenActions

Magix.DynamicEvent.ReInitializeOverriddenActions

Magix.MetaAction.ReInitializeOverriddenActions

Will Re-Initialize the Overridden Actions. Basically destroy all existing overrides, that comes from Actions, and re-run the overriding Initialization of whatever new values exists. By waiting with the remapping until you're done with your entire change of logic, you can rewire as many Actions as you wish, create new, and such. And when you're done, you can rewire all your Actions in one Swoop

GetFromCache

Magix.DynamicEvent.GetFromCache

Magix.Common.GetFromCache

Will append the given 'Key' Caching.Cache Page object into 'Value' if existing. If no 'Key' is given, but 'Contains' is given, it will return first element that contains 'Contains' string as 'Key'

ClearCache

Magix.DynamicEvent.ClearCache

Magix.Common.ClearCache

Will clear the Caching.Cache object associated with the Page object. Warning! Entire Cache will be emptied

RemoveFromCache

Magix.DynamicEvent.RemoveFromCache

Magix.Common.RemoveFromCache

Will remove the given 'Key' Caching.Cache Page object. If no 'Key' is given, but 'Contains' is given, it will sequentially remove everything that has 'Contains' as a part of its 'Key'

AddToCache

Magix.DynamicEvent.AddToCache

Magix.Common.AddToCache

Will append the given 'Value' into the Caching.Cache Page object with the given 'Key'

execute

Magix.DynamicEvent.execute

Magix.System.execute

Will play 'The-Last-Barfly.oga' if Node structure contains '{[Time].Value} >= 0200', otherwise play 'Cool-Breeze.oga'. Legal operators are ==, !=, >=, <=, >, <. All comparison is made by string comparison, alphabetically before, means less. The first if will show 'Hello Creator' if the 'Name' node contains 'Thomas Hansen' as its Value. The Expressions works such that you dereference your nodes, either by number, or name. Use 'root' to start traversing at root level, outer most parent. Use '../' to go upwards one level. Read more about Expressions and the Magix Turing Executor to understand the details of this Active Event and its possibilities

ClearOverriddenActions

Magix.DynamicEvent.ClearOverriddenActions

Magix.MetaAction.ClearOverriddenActions

Will Clear all Overridden Actions. Basically destroy all existing overrides, that comes from Actions

ScrollClientToTop

Magix.DynamicEvent.ScrollClientToTop

Magix.MetaView.ScrollClientToTop

Will scroll the browser all the way to the top. Useful for forms which are longer than the fold, when you submit them, and want to scroll to the top to display some message

CreateNodeFromMetaForm

Magix.DynamicEvent.CreateNodeFromMetaForm

Magix.MetaForms.CreateNodeFromMetaForm

Will put every Info field on your Meta Form into the Node structure underneath the 'Object' node

LoadMetaForm

Magix.DynamicEvent.LoadMetaForm

Magix.MetaForms.LoadMetaForm([Expression])

Will load the given 'MetaFormName' into the given 'Container' WebPart

ToggleCSSClassOfWidgetOrControl

Magix.DynamicEvent.ToggleCSSClassOfWidgetOrControl

Magix.Common.ToggleCSSClassOfWidgetOrControl

Will change the CSS class of the given Widget of Control referenced through its 'ID' parameter

ChangeAllPropertyValues

Magix.DynamicEvent.ChangeAllPropertyValues

Magix.Meta.ChangeAllPropertyValues

Will change the 'PropertyName' of every MetaObject of 'TypeName' to 'NewValue'

RollThroughEnumValuesForProperty

Magix.DynamicEvent.RollThroughEnumValuesForProperty

Magix.MetaTypes.RollThroughEnumValuesForProperty

Will roll through the 'Enums' values and change the value of the 'PropertyName' of the given 'ID' object

CreateNewMetaObject

Magix.DynamicEvent.CreateNewMetaObject

Magix.Meta.CreateNewMetaObject

Will create a new Meta Object with the 'Properties' properties and TypeName of 'TypeName' and parent Meta Object of ParentID

CreateLogEntry

Magix.DynamicEvent.CreateLogEntry

Magix.Core.Log

Will create a new Log entry in your Log. Copy and paste this Action to create your Own log objects

TransmitEventToExternalServer

Magix.DynamicEvent.TransmitEventToExternalServer

Magix.Core.TransmitEventToExternalServer

Will raise an external event to the given 'UrlEndPoint', raising the event with the 'EventName', with the parameters of 'Node'. The Remoting capacities of Magix, will by default store Authentication Cookies internally, associated with every unique base URL end-point, to keep your server Authenticated towards that specific server. Magix will store these cookies internally, and later re-transmit these Authentication cookies upon every future request, towards that same server. This means that you can for instance use the same state as you'd normally use in the Graphical rendering of your system. This facilitates for that you can do stuff such as first logging into the remote server raising LogIn or something, then to expect at that point, as a server to be authenticated forever towards that server, unless your Cookies are somehow destroyed, or your password somehow gets changed. Afterwards you can raise events which for instance requires you to be logged in, and expect them to be successfully executed

LogInUser

Magix.DynamicEvent.LogInUser

Magix.Core.LogInUser([Object])

Will login the given 'Username'/'Password' user. Either get these fields from your Form/View fields, or set them some other way

SaveNodeSerializedFromMetaForm

Magix.DynamicEvent.SaveNodeSerializedFromMetaForm

Magix.MetaForms.SaveNodeSerializedFromMetaForm

Will serialize the 'Object' node in your DataSource down do a graph depending upon how the Object node looks like

GetActiveTypeObjects

Magix.DynamicEvent.GetActiveTypeObjects

Magix.Common.GetActiveTypeObjects

Will return the Active Types of the given fully qualified name 'FullTypeName', as list within 'Objects'. Every node within 'Objects' will have an 'ID' property in addition to a 'Properties' property, which will contain every property you are requesting

DataBindForm

Magix.DynamicEvent.DataBindForm

Magix.MetaForms.DataBindForm

Will databind the form, which means it will evaluate every single expression, in every single property of every single widget on the Active Form such that every property that start with DataSource['xxx'] will become evaluated towards the DataSource of your WebForm using its expression. You can use expressions such as DataSource['Objects'][0]['Property'].Name or DataSource['FullTypeName'].Value, etc. The values of those nodes will be attempted converted into whatever the type of the property is according to the standard conversion logic in Magix

LoadObjectIntoForm

Magix.DynamicEvent.LoadObjectIntoForm

Magix.MetaView.LoadObjectIntoForm

Will assume a Meta Object is just loaded correctly into the Node structure somehow, and pass it on to any matching TypeName Single Views which will handle it and load up the values for the specific Meta Object into their specific editor controls

SetFocusToFirstTextBox

Magix.DynamicEvent.SetFocusToFirstTextBox

Magix.MetaView.SetFocusToFirstTextBox

Will set focus to the first control on the form if raised from within the same WebPart as the form it's trying to set focus to. Works only with Single Views. Explicitly set the 'OriginalWebPartID' to override to set focus to another WebPart's first TextBox on the Page

ValidateObjectPropertyMandatory

Magix.DynamicEvent.ValidateObjectPropertyMandatory

Magix.Common.ValidateObjectProperty

Will validate the given 'PropertyName' Value of the Active MetaView within the WebPart against the 'Type' parameter. Will throw an exception if validation fails with an error message back to user, meaning if you'd like to stop saving of a MetaObject due to validation failing, you'll need to have this Action before the SaveActiveForm Action. The 'Type' parameter can be either 'email', 'mandatory', 'number', 'full-name' or 'url' - Plus any of the gazillion plugins you might have other places in your system. For this Action the 'PropertyName' is 'Name', and the 'Type' is 'mandatory', which just means that if there's no value, or only spaces and non-alphabetical-characters, then it's considered empty, and an exception will be thrown

ValidateObjectPropertyFullName

Magix.DynamicEvent.ValidateObjectPropertyFullName

Magix.Common.ValidateObjectProperty

Will validate the given 'PropertyName' Value of the Active MetaView within the WebPart againsts the 'Type' parameter. Will throw an exception if validation fissles with an error message back to user, meaning if you'd like to stop saving of a MetaObject due to validation fissling, you'll need to have this Action before the SaveActiveForm Action. The 'Type' parameter can be either 'email', 'mandatory', 'number', 'full-name' or 'url' - Plus any of the gazillion plugins you might have other places in your system. For this Action the 'PropertyName' is 'Name', and the 'Type' is 'full-name', which means that at least two names need to be given, and the names will be normalized such as 'Hansen, Thomas' with the last name(s) first and all names automatically capitalized as a function of this Action. No automagic Capitalization will occur if Capital letters are found in the middle of a string though, such as 'McAngus'

ValidateObjectPropertyEmail

Magix.DynamicEvent.ValidateObjectPropertyEmail

Magix.Common.ValidateObjectProperty

Will validate the given 'PropertyName' Value of the Active MetaView within the WebPart againsts the 'Type' parameter. Will throw an exception if validation fissles with an error message back to user, meaning if you'd like to stop saving of a MetaObject due to validation fissling, you'll need to have this Action before the SaveActiveForm Action. The 'Type' parameter can be either 'email', 'mandatory', 'number', 'full-name' or 'url' - Plus any of the gazillion plugins you might have other places in your system. For this Action the 'PropertyName' is 'Email', and the 'Type' is 'email'. Please note that if you wish to allow for either a valid email address, or an empty value, you need to pass in 'AcceptNull' being true

ValidateObjectPropertyNumber

Magix.DynamicEvent.ValidateObjectPropertyNumber

Magix.Common.ValidateObjectProperty

Will validate the given 'PropertyName' Value of the Active MetaView within the WebPart againsts the 'Type' parameter. Will throw an exception if validation fissles with an error message back to user, meaning if you'd like to stop saving of a MetaObject due to validation fissling, you'll need to have this Action before the SaveActiveForm Action. The 'Type' parameter can be either 'email', 'mandatory', 'number', 'full-name' or 'url' - Plus any of the gazillion plugins you might have other places in your system. For this Action the 'PropertyName' is 'Age', and the 'Type' is 'number'. Please note that if you wish to allow for either a valid number address, or an empty value, you need to pass in 'AcceptNull' being true. The valid characters in a 'number' are; '0123456789,., '.

CreateGallery

Magix.DynamicEvent.CreateGallery

Magix.Common.CreateGallery

Will create a Gallery object from the given 'Files' list within the 'Folder'

IncludeCSSFile

Magix.DynamicEvent.IncludeCSSFile

Magix.Core.AddCustomCssFile

Will include a CSS file onto the page, even in an Ajax Callback if you wish. Change the 'CSSFile' parameter to choose which CSS file you'd like to include

CreateQRCode

Magix.DynamicEvent.CreateQRCode

Magix.QRCodes.CreateQRCode

Will create a QR Code with the given 'FileName' path and filename, which should end with .png. The QR Code will point to the given 'URL', and it will use the textures of 'BGImage' and 'FGImage' to render the code. The QR Code will have 'RoundedCorners' radius of rounded corners, and it will be 'AntiPixelated', and have the descriptive text of 'Text'. The QR Code will use 'Scale' number of pixels per square to render

PlaySound

Magix.DynamicEvent.PlaySound

Magix.Core.PlaySound

Will play the given 'File' sound. If the sample sound file doesn't work, you've probably got something wrong with the setup of your Web Server. Make sure the file extension .oga is associated with the MIME type of audio/ogg. PS! For the record; Cool-Breeze.oga and The-Last-Barfly.oga are both songs composed by our CTO Thomas Hansen. They are both performed by Thomas Hansen and his wife Inger Hoeoeg, and are to be considered licensed to you under the terms of Creative Commons Attribution-ShareAlike 3.0

PauseSound

Magix.DynamicEvent.PauseSound

Magix.Core.PauseSound

Stops Any sound or music currently being played

ResumeSound

Magix.DynamicEvent.ResumeSound

Magix.Core.ResumeSound

Resumes any sound or music previously being halted through 'PauseSound' or other similar mechanisms. PS! Will throw exception if no sounds have been played, and hence no resuming can occur in any ways

EmptyAndClearActiveForm

Magix.DynamicEvent.EmptyAndClearActiveForm

Magix.MetaView.EmptyForm

Will empty the currently active Editable Form. Will determine which form raised the event originally, and explicitly empty that form only. Useful for things such as 'Clear Buttons' and such ...

RedirectClient

Magix.DynamicEvent.RedirectClient

Magix.Common.RedirectClient

Will redirect the client's browser to the given URL parameter

PutGETParameterIntoDataSource

Magix.DynamicEvent.PutGETParameterIntoDataSource

Magix.Common.PutGETParameterIntoDataSource

Will get the given parameter from the URL string and put it into the given node

Transform

Magix.DynamicEvent.Transform

Magix.Common.Transform

Will transform the given parameter node immutably [will not change the source] and return as a new Node collection according to the expression(s) in the 'Expression' parameter

GetMetaObjectGraph

Magix.DynamicEvent.GetMetaObjectGraph

Magix.Common.GetMetaObjectGraph

Will return the entire Graph [warning, might be HUGE!] for the MetaObject given, with all its Children Meta Objects too within the 'Object' node of your parameter

ViewCarsInPopup

Magix.Demo.ViewCarsInPopup

Magix.MetaView.ViewMetaViewMultiMode

Will load up any Magix.Demo.Car objects into a MultiView form, within a popup window, using the Magix.Demo.ImportCars MetaView. If there are no items you can run the 'Magix.Demo.ImportCarsCSVFile' action, which by default will point towards a CSV file which contains 'Cars' data

ImportCarsCSVFile

Magix.Demo.ImportCarsCSVFile

Magix.Common.ImportCSVFile

Will import the given 'FileName' from the given 'Folder' and transform to MetaObjects using the given 'MetaViewName'

ShowMessage

Magix.DynamicEvent.ShowMessage

Magix.Core.ShowMessage

Will show a 'Message Box' to the User. If you add 'IsError' to true, the message box will be in error mode, meaning red probably, signifying a an error

TurnOnDebugging

Magix.DynamicEvent.TurnOnDebugging

Magix.Common.SetSessionVariable

Will turn on 'Debugging', meaning you'll have a wire-grid covering your screen to see the 40x18 pixel 'grid-lock', plus you'll also get to see every single Action ever raised on the server shown in an 'Action Stack Trace' Window. This only affects your session, meaning it should be safe to do in production to track down errors and such in live software ...

TurnOffDebugging

Magix.DynamicEvent.TurnOffDebugging

Magix.Common.SetSessionVariable

Will turn `_OFF_` 'Debugging', meaning you'll no longer have a wire-grid covering your screen, plus the stack tracing of actions on the server will disappear. Only affects your session, and no other logged on users ability to see debugging information ...

ViewMetaViewMultiMode

Magix.DynamicEvent.ViewMetaViewMultiMode

Magix.MetaView.ViewMetaViewMultiMode

Will load a grid of all Meta Objects of type already loaded in current activating WebPart. If you want to load a specific type, then you can override the type being loaded by adding 'MetaViewTypeName' as a parameter, containing the name of the view

SendEmail

Magix.DynamicEvent.SendEmail

Magix.Common.SendEmail

Will send an email to the 'To' address

SendEmailFromForm

Magix.DynamicEvent.SendEmailFromForm

Magix.Common.SendEmailFromForm

Helper to further assist on creating 'Send Email Forms'. Will send an email with the given 'Body' and 'Header' to the email address within the 'Email' field of the Active SingleView Form View. It will also replace every instance of [x] with the MetaView's property with the same named value. Meaning if you've got a MetaView property called 'Name' it will replace all occurrences of [Name] with the value from the MetaObject's property of 'Name' in both the body and the header. PS! This Action can be seen in 'Action' in the 'Magix.Demo.SendEmail' MetaView

ReplaceStringValue

Magix.DynamicEvent.ReplaceStringValue

Magix.Common.ReplaceStringValue

Will transform every entity of 'OldString' found in 'Source' into the contents of 'NewString' and return as a 'Result', output node. You can also use 'SourceNode', 'NewStringNode' and 'ResultNode' to override which Nodes to read values and store result into

GetSingleMetaObject

Magix.DynamicEvent.GetSingleMetaObject

Magix.Common.GetSingleMetaObject

Will put every property from the given Meta Object, into the given Node, with the name/value pair as the node name/value parts, assuming they're all strings. Copy this Action, and make sure you `_CHANGE_` its ID towards pointing to the ID of a real existing Meta Object, to fill in values from one of your Meta Objects into a Node, maybe before sending the node into another even, using `MultiActions` or something ...

ReplaceStringLiteral

Magix.DynamicEvent.ReplaceStringLiteral

Magix.Common.ReplaceStringLiteral

Expects three parameters, 'Replace', 'Replacement' and 'Source'. Will replace every occurrence of 'Replace' in the 'Source' Node Expression with either the expression or the value in the 'Replacement' parameter

GetMetaObjects

Magix.DynamicEvent.GetMetaObjects

Magix.MetaObjects.GetMetaObjects

Will return from 'Start' to 'End' MetaObjects sorted according to newest first of Type Name 'MetaTypeName'. If you set 'Ascending' to false, it will sort according to oldest first. Modify the 'Start' and 'End' parameters to change your data extraction

GetRatingObjects

Magix.Demo.GetRatingObjects

Magix.MetaObjects.GetMetaObjects

Will return the 5 newest `Magix.Demo.Rating` Meta Objects

GetNextMetaObjects

Magix.DynamicEvent.GetNextMetaObjects

Magix.MetaObjects.GetNextMetaObjects

Will return from 'Start' to 'End' MetaObjects sorted according to newest first of Type Name 'MetaTypeName'. But only AFTER both 'Start' and 'End' have been incremented by 'End' - 'Start'. If you set 'Ascending' to false, it will sort according to oldest first. Modify the 'Start' and 'End' parameters to change your data extraction. Useful for chaining together with its Previous counterpart, and its get specific extraction counterpart

GetPreviousMetaObjects

Magix.DynamicEvent.GetPreviousMetaObjects

Magix.MetaObjects.GetPreviousMetaObjects

Will return from 'Start' to 'End' MetaObjects sorted according to newest first of Type Name 'MetaTypeName'. But only AFTER both 'Start' and 'End' have been decremented by 'End' - 'Start'. If you set 'Ascending' to false, it will sort according to oldest first. Modify the 'Start' and 'End' parameters to change your data extraction. Useful for chaining together with its Previous counterpart, and its get specific extraction counterpart

SetMetaObjectValue

Magix.DynamicEvent.SetMetaObjectValue

Magix.MetaType.SetMetaObjectValue

Will set the value of the given MetaObject [ID] to the Value of your 'Value' node at the 'Name' property.

SerializeActiveSingleViewForm

Magix.DynamicEvent.SerializeActiveSingleViewForm

Magix.MetaView.SerializeSingleViewForm

Will find the Form that raised the current eventchain, and query it to put all its data flat out into the current Node structure with the Name/Value as the Node Name/Value pair. Note; Can mostly only be raised from within a Single View MetaView, unless you're doing other overrides

LoadSignatureModule

Magix.DynamicEvent.LoadSignatureModule

Magix.Common.LoadSignatureForCurrentMetaObject

Will load the Signature Module in whatever container its being raised from. And set the 'Value' property of the given MetaObject Column Property Name to the signature signed on the Signature module ...

ExportMetaView2CSV

Magix.DynamicEvent.ExportMetaView2CSV

Magix.Common.ExportMetaView2CSV

Will render the 'Currently Viewed' MetaView into a CSV file [Microsoft Excel or Apple Numbers etc] and redirect the users client [Web Browser] to the newly rendered CSV file. 'Currently Viewed' meaning the view that contained the control that initiated this Action somehow. PS! If you explicitly create a 'MetaViewName' parameter, and set its name to another MetaView, then that MetaView will be rendered instead

3. CompressionModule

Magix.Brix.ApplicationPool.CompressionModule

Description:

Level3: Helper class for implementing GZIP-Compression of resources and pages. PS! Thanx and credits goes to Mads Kristiansen for sharing this on his blog. Google 'CompressionModule Mads Kristiansen' if more background information is needed about this piece of code

4. MainWebPage

Magix.Brix.ApplicationPool.MainWebPage

Description:

Level4: Your 'Application Pool', meaning the 'world' where all your 'components' lives. Nothing really to see here, this should just 'work'. But are here for reference reasons

5. Magix_PageStatePersister

Magix.Brix.ApplicationPool.Magix_PageStatePersister

Description:

Level4: Implements serialization of ViewState into the database on the server side

Methods

public override void Load()

Level3: Loads Viewstate from database

public override void Save()

Level3: Saves Viewstate to database

6. ActiveTypeCached-g

Magix.Brix.Components.ActiveTypeCached-g

Description:

Level3: Helper class for going through cache to retrieve Active Types. Will cache most queries

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

```
public new T SelectFirst(params Criteria[])
```

Level3: Returns the first object from your data storage which are true for the given criterias. Pass nothing () if no criterias are needed.

```
public new IEnumerable( T ) Select(params Criteria[])
```

Level3: Returns all objects from your data storage that matches the given criterias. Pass nothing () if you want all objects regardless of any values or criterias.

```
public override void Save()
```

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

```
public override void Delete()
```

Level3: Deletes the object from your data storage.

7. TextAreaEdit

Magix.Brix.Components.TextAreaEdit

Description:

Level4: Basically every single instance you see in Magix which is of type 'InPlaceEditable' is an instance of this component. Allows for viewing textual based information, while at the same time, this tet can be 'clicked' which will change it into a text box, in which you can change its value. Often displayed with a pen next to it

Basetypes

Magix.UX.Widgets.Panel

Events

public EventHandler TextChanged

Level4: Text was changed by end user

public EventHandler DisplayTextBox

Level4: Raised when TextBox is about to become displayed

Properties

public string Text

Level4: The actual text property of the control

public int TextLength

Level4: Max length before 'clipping' will occur

8. Module

Magix.Brix.Components.Module

Description:

Level4: Baseclass for the 'Edit Objects' parts of the Grid system in Magix. Not meant for directly consuming through this class

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

```
[ActiveEvent(Name = "Magix.Core.UpdateGrids")]  
protected void Magix_Core_UpdateGrids(object sender, EventArgs e)
```

Level4: Handled to make sure we update ours too, since object might be changed some of its properties

```
[ActiveEvent(Name = "DBAdmin.Data.ChangeSimplePropertyValue")]  
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, EventArgs e)
```

Level4: Checking to see if it may be us, and if so, re-databind

```
[ActiveEvent(Name = "Magix.Core.RefreshWindowContent")]  
protected void RefreshWindowContent(object sender, EventArgs e)
```

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

9. ListModule

Magix.Brix.Components.ListModule

Description:

Level4: Implements most of the logic for our DBAdmin module and basically every single Grid we have in Magix. This class is the foundation for probably most Magix screens you've ever seen, since every Grid within the system, virtually, is built using this class. Have so many bells and whistles, it could probably need its own book, not to mention some serious refactoring. Till at least the refactoring parts are over, I think I'll resist the temptation of going 'over the board' in documenting this, since first of all everything will change later, and you shouldn't fiddle too much directly with this bugger yourself. There's also PLENTY code samples around ANYWHERE in magix of usage of this class, so have fun :)

Basetypes

Magix.Brix.Components.Module

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

protected abstract void DataBindDone()

Level4: Called when databinding are done

[ActiveEvent(Name = "DBAdmin.Grid.GetActiveFilters")]

protected void DBAdmin_Grid_GetActiveFilters(object sender, EventArgs e)

Level2: Will return the filter criteria for the grid ...

```
[ActiveEvent(Name = "DBAdmin.Grid.SetActiveRow")]  
protected void DBAdmin_Grid_SetActiveRow(object sender, ActiveEventArgs e)
```

Level2: Will set the Active Row of the grid, if the 'FullTypeName' is correct

Properties

```
protected abstract ControlTableParent
```

Level4: The control being our 'table element'

10. Clipboard_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.ClipBoard.ClipBoard_Controller

Description:

Level2: Encapsulates the logic needed for manipulating the clipboard, add items to it, and paste items to the Clipboard. Notice that by overriding some of these events, and have them go to other servers, you can copy and paste any clipboard items from one server to the other

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]

protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Will return the menu items needed to fire up 'View Clipboard'

[ActiveEvent(Name = "Magix.ClipBoard.CopyNode")]

protected void Magix_ClipBoard_CopyNode(object sender, EventArgs e)

Level2: Will take the incoming 'ClipboardNode' and append it into the current user's Session object

[ActiveEvent(Name = "Magix.ClipBoard.ShowClipboard")]

protected void Magix_ClipBoard_ShowClipboard(object sender, EventArgs e)

Level2: Will load the clipboard module into the floater Container

[ActiveEvent(Name = "Magix.ClipBoard.RemoveItemFromClipboard")]

protected void Magix_ClipBoard_RemoveItemFromClipboard(object sender, EventArgs e)

Level2: Will erase the given 'ID' item from the Clipboard

[ActiveEvent(Name = "Magix.ClipBoard.PrepareToPasteNode")]

protected void Magix_ClipBoard_PrepareToPasteNode(object sender, EventArgs e)

Level2; Will prepare to send the PasteItem event. Expects the 'ID' to point to a ClipBoard Item ID

[ActiveEvent(Name = "Magix.ClipBoard.CloseClipBoard")]

protected void Magix_ClipBoard_CloseClipBoard(object sender, EventArgs e)

Level2: Closes the Clipboard Module, if it is up

11. Common_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Common.Common_Controller

Description:

Level2: Contains some common helpers and logic that doesn't belong at one particular place. For instance loading of a color picker dialogue

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.PickColorOrImage")]

protected void Magix_Core_PickColorOrImage(object sender, EventArgs e)

Level2: Will load up a Color Picker dialogue such that the end user can pick a color or image for using as e.g. background color/textures. See the PickColorOrImage module for details about every configurable parameter

[ActiveEvent(Name = "Magix.Core.FilePicker.SelectFileInsteadOfColor")]

protected void Magix_Core_FilePicker_SelectFileInsteadOfColor(object sender, EventArgs e)

Level2: Will launch the File Explorer to allow the end user to pick an image for rendering as e.g. background textures for widgets etc

12. DBAdmin_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.DBAdmin.DBAdmin_Controller

Description:

Level2: Contains the logic for the DBAdmin module(s), which is also the Grid/CRUD-foundation system in Magix. Contains many useful methods and ActiveEvents for displaying either Grids or editing Single Instances of Objects. Can react 100% transparently on ActiveTypes. Has support for Meta Types, meaning types where you're more in 'control', but must do more of the 'arm wrestling directly'

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "DBAdmin.Form.ViewClasses")]

protected void DBAdmin_Form_ViewClasses(object sender, EventArgs e)

Level2: Loads up the DBAdmin BrowserClasses interface. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

[ActiveEvent(Name = "DBAdmin.Data.GetClassHierarchy")]

protected void DBAdmin_Data_GetClassHierarchy(object sender, EventArgs e)

Level2: Will return the Class Hierarchy of all ActiveTypes within the system in a tree hierarchy, according to namespace


```
[ActiveEvent(Name = "DBAdmin.Form.ViewClass")]
protected void DBAdmin_Form_ViewClass(object sender, EventArgs e)
```

Level2: Will show all the objects of type 'FullTypeName'. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

```
[ActiveEvent(Name = "DBAdmin.Data.GetContentsOfClass")]
protected void DBAdmin_Data_GetContentsOfClass(object sender, EventArgs e)
```

Level2: Will return a range of objects type 'FullTypeName' depending upon the 'Start' and 'End' parameter. Requires a 'FullTypeName' parameter, and 'Start' and 'End' parameters

```
[ActiveEvent(Name = "DBAdmin.Data.ChangeSimplePropertyValue")]
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, EventArgs e)
```

Level2: Will change a 'Simple Property Value' [property of some sort belonging to the object, e.g. a DateTime/Birthday property] Needs 'FullTypeName', 'ID', 'PropertyName', 'NewValue' to work. NewValue must be of type of property.

```
[ActiveEvent(Name = "DBAdmin.Form.ViewListOrComplexPropertyValue")]
protected void DBAdmin_Form_ViewListOrComplexPropertyValue(object sender,
    EventArgs e)
```

Level2: Will show either a Child object or a List of children depending upon the 'IsList' parameter. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

```
[ActiveEvent(Name = "DBAdmin.Data.GetListFromObject")]
protected void DBAdmin_UpdateComplexValue(object sender, EventArgs e)
```

Level2: Returns a Range of Child Objects belonging to the 'ParentID'. Needs 'ParentFullTypeName', 'ParentPropertyName', 'Start', 'End' and 'ParentID' to function

```
[ActiveEvent(Name = "DBAdmin.Data.GetObject")]
protected void DBAdmin_Data_GetObject(object sender, EventArgs e)
```

Level2: Will return the object with the given 'ID' being of 'FullTypeName' as a Value/Key pair

```
[ActiveEvent(Name = "DBAdmin.Data.GetObjectFromParentProperty")]
protected void DBAdmin_Data_GetObjectFromParentProperty(object sender, EventArgs e)
```

Level2: Will return a single instance of a complex [ActiveType normally, unless 'meta'] child object

```
[ActiveEvent(Name = "DBAdmin.Form.ViewComplexObject")]
protected void DBAdmin_ShowComplexObject(object sender, EventArgs e)
```

Level2: Will show one Complex object with the 'ID' and 'FullTypeName'. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

```
[ActiveEvent(Name = "DBAdmin.Form.ConfigureFilterForColumn")]
protected void DBAdmin_Form_ConfigureFilterForColumn(object sender, EventArgs e)
```

Level2: Will open up a 'Configure Filter' dialogue from which the user can change, edit or remove any existing filters or create new ones

```
[ActiveEvent(Name = "DBAdmin.Form.ShowAddRemoveColumns")]
protected void DBAdmin_Form_ShowAddRemoveColumns(object sender, EventArgs e)
```

Level2: Will load up 'Configure Columns to View form' to end user

```
[ActiveEvent(Name = "DBAdmin.Data.ChangeVisibilityOfColumn")]
protected void DBAdmin_Data_ChangeVisibilityOfColumn(object sender, EventArgs e)
```

Level2: Changes the visibility setting of a specific Column for a specific type

```
[ActiveEvent(Name = "DBAdmin.Data.DeleteObject")]
protected void DBAdmin_Data_DeleteObject(object sender, EventArgs e)
```

Level2: Will delete the given 'ID' ActiveType within the 'FullTypeName' namespace/name after user has been asked to confirm deletion

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Data_ComplexInstanceDeletedConfirmed(object sender,
    EventArgs e)
```

Level2: Default implementation of 'deletion of object was confirmed by user' logic

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedNotConfirmed")]
protected void DBAdmin_Data_ComplexInstanceDeletedNotConfirmed(object sender,
    EventArgs e)
```

Level2: Flushes the Container containing the MessageBox

```
[ActiveEvent(Name = "DBAdmin.Common.CreateObject")]
protected void DBAdmin_Common_CreateObject(object sender, EventArgs e)
```

Level2: Will create a new object of type 'FullTypeName'

```
[ActiveEvent(Name = "DBAdmin.Common.CreateObjectAsChild")]
protected void DBAdmin_Common_CreateObjectAsChild(object sender, EventArgs e)
```

Level2: Will create a new object of type 'ParentFullTypeName' and append it to the 'ParentID' 'ParentPropertyName' property which must be of type 'FullTypeName'

```
[ActiveEvent(Name = "DBAdmin.Form.AppendObject")]
protected void DBAdmin_Form_AppendObject(object sender, EventArgs e)
```

Level2: Will show a list of objects of type 'FullTypeName' and allow the user to pick one to append into 'ParentID' 'ParentPropertyName' with the given 'ParentFullTypeName'. If 'ReUseNode' is true, then you can position the module. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

```
[ActiveEvent(Name = "DBAdmin.Data.AppendObjectToParentPropertyList")]
protected void DBAdmin_Data_AppendObjectToParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Will append an object to a list of objects in 'ParentID' ParentPropertyName collection and save the 'ParentID' object

```
[ActiveEvent(Name = "DBAdmin.Data.ChangeObjectReference")]
protected void DBAdmin_Data_ChangeObjectReference(object sender, ActiveEventArgs e)
```

Level2: Will change a single instance object reference between 'ParentID' and 'ID' in the 'ParentPropertyName' of 'ParentFullTypeName'. Flushes child container

```
[ActiveEvent(Name = "DBAdmin.Form.RemoveObjectFromParentPropertyList")]
protected void DBAdmin_Form_RemoveObjectFromParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Removes a referenced object without deleting it [taking it out of its parent collection] Notice that if the Parent object is the 'Owner' of the object, it may still be deleted. Will ask for confirmation from end user before operation is performed

```
[ActiveEvent(Name = "DBAdmin.Form.RemoveObjectFromParentPropertyList- Confirmed")]
protected void DBAdmin_Data_RemoveObjectFromParentPropertyList(object sender,
ActiveEventArgs e)
```

Level2: Removes an object out of its 'ParentID' 'ParentPropertyName' collection of type 'ParentFullTypeName'

```
[ActiveEvent(Name = "DBAdmin.Form.ChangeObject")]
protected void DBAdmin_Form_ChangeObject(object sender, ActiveEventArgs e)
```

Level2: Will show the 'Change Single-Object Reference' form to the end user. If 'ReUseNode' is true, then Module positioning parameters can be applied. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

```
[ActiveEvent(Name = "DBAdmin.Data.RemoveObject")]  
protected void DBAdmin_Form_RemoveObject(object sender, EventArgs e)
```

Level2: Removes a single-object reference from the 'ParentID' object

```
[ActiveEvent(Name = "DBAdmin.Data.GetFilter")]  
protected void DBAdmin_Data_GetFilter(object sender, EventArgs e)
```

Level2: Returns the filters for different columns in the Grid system

```
[ActiveEvent(Name = "DBAdmin.Data.SetFilter")]  
protected void DBAdmin_Data_SetFilter(object sender, EventArgs e)
```

Level2: Changes the filter for a specific 'Key'/'Value' for a specific type

```
[ActiveEvent(Name = "DBAdmin.Grid.DeleteAllObjects")]  
protected void DBAdmin_Grid_DeleteAllObjects(object sender, EventArgs e)
```

Level2: Will delete all objects within the range of the given 'Criteria' collection. But only after the user has been warned, and asked to confirm

```
[ActiveEvent(Name = "DBAdmin.Grid.DeleteAllObjectsNotConfirmed")]  
protected void DBAdmin_Grid_DeleteAllObjectsNotConfirmed(object sender, EventArgs e)
```

Level2: Just closes our Dialog Box ...

```
[ActiveEvent(Name = "DBAdmin.Grid.DeleteAllObjectsConfirmed")]  
protected void DBAdmin_Grid_DeleteAllObjectsConfirmed(object sender, EventArgs e)
```

Level2: Will delete all objects within the range of the given 'Criteria' collection

13. ColumnTypesPlugins_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.DBAdmin.ColumnTypesPlugins_Controller

Description:

Level3: Contains some template columns for the Grid system for you to use in your own Grids

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.DataPlugins.GetTemplateColumns.CheckBox")]
protected void Magix_DataPlugins_GetTemplateColumns_CheckBox(object sender,
ActiveEventArgs e)

Level3: Creates a CheckBox type of column for the Grid System

14. Distributor_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Distributor.Distributor_Controller

Description:

Level2: Helper class for serializing and transmitting back and forth between other servers

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender, ActiveEventArgs e)

Level2: Sink for Dashboard Plugins

[ActiveEvent(Name = "Magix.Publishing.ViewServerRequests")]
protected void Magix_Publishing_ViewServerRequests(object sender, ActiveEventArgs e)

Level2: Shows the Open and Pending Server Requests you have

[ActiveEvent(Name = "Magix.Core.PostHTTPRequest")]
protected void Magix_Core_PostHTTPRequest(object sender, ActiveEventArgs e)

Level2: Will check to see if this is a WebService call, if not, it will forward the event to the default implementation. When Magix fires an external event, the external event will transfer execution to the receiving party as an HTTP request. This event would normally be raised when such an event occurs. This event will expect an 'event', which will be the Active Event raised, and 'params' will become the parameters passed into the Active Event. Both 'event' and 'params' are fetched from the HTTP Request Parameters collection

```
[ActiveEvent(Name = "Magix.Core.TransmitEventToExternalServer")]  
protected void Magix_Core_TransmitEventToExternalServer(object sender, ActiveEventArgs e)
```

Level2: Will raise an event to the externally given 'UrlEndPoint', which should be an HTTP end point which can take 'event' and 'params' as HTTP POST parameters. 'EventName' is the name of the 'event' to raise externally, 'Node' will be transmitted as its 'params'. An HTTP POST request will be created with 'event' [string] will be made from 'EventName', and 'params' [JSON] will be made from 'Node'. The endpoint URL is expected to be found in 'UrlEndPoint'. Please notice that most servers would by default have to explicitly grant you access to raise external events, meaning if you raise an event to another server, the admin of that server needs to allow you to explicitly raise that specific event within his server

15. Documentation_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Documentation.Documentation_Controller

Description:

Level2: Contains the logic for our 'Class Browser' which can browse all the classes in the system and make some changes to them by enabling them and disabling them by either overriding specific events or by disabling entire controllers or modules all together

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]
protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, ActiveEventArgs e)

Level2: Will populate the Desktop

[ActiveEvent(Name = "Magix.Documentation.LaunchHeka")]
protected void Magix_Documentation_LaunchHeka(object sender, ActiveEventArgs e)

Level2: Will return the menu items needed to start the class browser

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, ActiveEventArgs e)

Level2: Will return the menu items needed to start the class browser

[ActiveEvent(Name = "Magix.Core.GetDocumentationForActiveEvent")]
protected void Magix_Core_GetDocumentationForActiveEvent(object sender, ActiveEventArgs e)

Level2: Sink for returning documentation for Active Event. Pass in 'ActiveEventName' and 'Result' will contain the documentation, concatenated from all its Event Handlers

```
[ActiveEvent(Name = "Magix.MetaType.ViewDoxygenFiles")]
protected void Magix_MetaType_ViewDoxygenFiles(object sender, EventArgs e)
```

Level2: Gets every namespace and class into a tree hierarchy and loads up the Tree module to display it

```
[ActiveEvent(Name = "Magix.Doxygen.Nei!Det_E_Boka_Mi ...!")]
protected void Magix_Doxygen_Give_Me_My_Bok_Mann(object sender, EventArgs e)
```

Will generate the PDF containing the entire documentation, plus potential plugins documentation features, such as the MetaAction and MetaView system etc. Will redirect the client to download the PDF file once generated.

```
[ActiveEvent(Name = "Magix.Doxygen.RollLevel")]
protected void Magix_Doxygen_RollLevel(object sender, EventArgs e)
```

Rolls level on documentation advanceness one up, til 4, and then back to zero. Reloads Doxygen Documentation class/namespace Tree afterwards

```
[ActiveEvent(Name = "Magix.Doxygen.ViewNamespaceOrClass")]
protected void Magix_Doxygen_ViewNamespaceOrClass(object sender, EventArgs e)
```

Level2: Expects a SelectedItemID which should point to either a Namespace or a Class, and will show this namespace/class features. Such as which dll(s) implements the namespace/class, if a class, which methods it has etc

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, EventArgs e)
```

Level2: Handled to make sure we handle the "Documentation_Controller-META" type for the Grid system so that we can see our DLLs

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, EventArgs e)
```

Level2: Handled to make sure we handle the "Documentation_Controller-META" type for the Grid system so that we can see our DLLs

16. Email_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Email.Email_Controller

Description:

Level2: Implements logic for sending email using SMTP through the .Net classes

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Async = true, Name = "Magix.Core.SendEmail")]

protected void Magix_Core_SendEmail(object sender, EventArgs e)

Level2: Will send an Email, using the SMPT settings from web.config, with the given 'Header' and 'Body' to the list in the 'EmailAddresses' parameter from the 'AdminEmail ' and 'AdminEmailFrom' parameters

17. FileExplorer_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.FileExplorer.FileExplorer_Controller

Description:

Level2: Contains Logic for file explorer. The file explorer is a component where you can browse the file folder system on your web server. Such as editing CSS files, uploading images and such. PS! To use this Module it is IMPERATIVE that you've given the 'NETWORK SERVICE' account 'Full Access' to at the very least the 'media/' folder, or whatever folder you plan to use the explorer at on your web server

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.FileExplorer.LaunchExplorer")]

protected void Magix_FileExplorer_LaunchExplorer(object sender, ActiveEventArgs e)

Level2: Use this method to launch the file explorer. You can override any parameter you wish. The default file filter is; "*.png;*.jpeg;*.jpg;*.gif;" but can be changed through the "Filter" parameter. If you add up CSS, and set "CanCreateNewCssFile" to true, then CSS files can both be edited and created on the fly. If you set "IsSelect", then the end user can "select" files, which upon selection is done will raise the 'SelectEvent'. To position your Module, use these parameters: 'Width', 'Height', 'Top', 'Push', 'Container', 'Last' and 'Overflow'. Top and push allows you to add margin from the top, while last is needed if the module covers the whole viewport in its width. Overflow allows you to have an undefined and relative height depending upon the contents height. All Units are integers, except last and overflow which are booleans. Total width of CSS Grid system is 24 columns and infinite height. Every Cell is 30x18 pixels wide, and has 10 pixels of right margin.

[ActiveEvent(Name = "Magix.FileExplorer.GetFilesFromFolder")]

protected void Magix_FileExplorer_GetFilesFromFolder(object sender, ActiveEventArgs e)

Level2: Will retrieve the 'FolderToOpen' folder's files back to caller

```
[ActiveEvent(Name = "Magix.FileExplorer.EditAsciiFile")]  
protected void Magix_FileExplorer_EditAsciiFile(object sender, ActiveEventArgs e)
```

Level2: Will open up EitAsciiFile Notepad-Type Editor with the given 'File' for editing. Intended to allow editing of CSS files and other types of ASCII files. 'File' should be the relative path of a file from the root of your application

```
[ActiveEvent(Name = "Magix.FileExplorer.FileSelected")]  
protected void Magix_FileExplorer_FileSelected(object sender, ActiveEventArgs e)
```

Level2: Will retrieve the properties for the file, such as disc size, width/height if image, etc

```
[ActiveEvent(Name = "Magix.FileExplorer.ChangeFileName")]  
protected void Magix_FileExplorer_ChangeFileName(object sender, ActiveEventArgs e)
```

Level2: Changes the name of the file with 'OldName' to 'NewName' within the given 'Folder'. The file ending cannot be changed

```
[ActiveEvent(Name = "Magix.FileExplorer.DeleteFile")]  
protected void Magix_FileExplorer_DeleteFile(object sender, ActiveEventArgs e)
```

Level2: Will delete the 'File' within the 'Folder' of the system

18. GalleryPlugin_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.GalleryPlugin.GalleryPlugin_Controller

Description:

Level2: Gallery plugin for the Gallery modules. Also serves as a nice vantage point for creating your own plugins to Magix and the publishing system

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Handled to override the 'DBAdmin.Common.ComplexInstanceDeletedConfirmed' event

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]

protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Loads the Administrator SlidingMenu for our GalleryPlugin injected

[ActiveEvent(Name = "Magix.Publishing.Gallery.EditGalleries")]

protected void Magix_Publishing_Gallery_EditGalleries(object sender, EventArgs e)

Level2: Loads up the grid for editing all galleries

[ActiveEvent(Name = "Magix.Gallery.GetUserTemplateColumn")]

protected void Magix_Gallery_GetUserTemplateColumn(object sender, EventArgs e)

Level2: Creates a Label with the username of the Gallery and returns back to caller

```
[ActiveEvent(Name = "Magix.Publishing.Gallery.EditGallery")]
protected void Magix_Publishing_Gallery_EditGallery(object sender, EventArgs e)
```

Level2: Edits the given 'ID' Gallery

```
[ActiveEvent(Name = "Magix.Publishing.Gallery.GetAllGalleries")]
protected void Magix_Publishing_Gallery_GetAllGalleries(object sender, EventArgs e)
```

Level2: Sink for getting all Galleries

```
[ActiveEvent(Name = "Magix.Publishing.Gallery.OpenGalleryInCurrentContainer")]
protected void Magix_Publishing_Gallery_OpenGalleryInCurrentContainer(object sender,
    EventArgs e)
```

Level2: Will open up the given 'ID' Gallery in the container from within the WebPart which the Active Event was raised from within

```
[ActiveEvent(Name = "Magix.Gallery.FileWasAppendedToGallery")]
protected void Magix_Gallery_FileWasAppendedToGallery(object sender, EventArgs e)
```

Level3: File was appended to gallery, saving the file locally, within the folder of the Gallery, and attaching the image to the existing Gallery

```
[ActiveEvent(Name = "Magix.Gallery.AppendButtonClicked")]
protected void Magix_Gallery_AppendButtonClicked(object sender, EventArgs e)
```

Level2: Append Image button was clicked for specific gallery

```
[ActiveEvent(Name = "Magix.Publishing.AppendImageToGallery")]
protected void Magix_Publishing_AppendImageToGallery(object sender, EventArgs e)
```

Level2: New Image file was added to Gallery. 'Folder', 'FileName' and 'Params/GalleryID' needs to be specified

```
[ActiveEvent(Name = "Magix.Gallery.GetGalleryData")]
protected void Magix_Gallery_GetGalleryData(object sender, EventArgs e)
```

Level3: Will return all images from the given 'GalleryName' Gallery as 'Images'

```
[ActiveEvent(Name = "Magix.Common.CreateGallery")]  
protected void Magix_Common_CreateGallery(object sender, EventArgs e)
```

Level2: Will create a Gallery component from a list of images ['Files'] within the given 'Folder' and return its id as 'ID'

```
[ActiveEvent(Name = "Magix.Gallery.GetTemplateColumnSelectGallery")]  
protected void Magix_Gallery_GetTemplateColumnSelectGallery(object sender, EventArgs  
e)
```

Level3: Will return a SelectList with all the Galleries back to caller

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed- Override")]  
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed_Override(object sender,  
EventArgs e)
```

Level2: Overridden to clear from content 4 and out ...

19. Logger_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Logger.Logger_Controller

Description:

Level2: Contains logic for Logging things

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.SendEmail")]
protected void Magix_Core_SendEmail(object sender, ActiveEventArgs e)

Level2: Logs every email sent out of Magix

[ActiveEvent(Name = "Magix.Core.ShowMessage")]
protected void Magix_Core_ShowMessage(object sender, ActiveEventArgs e)

Level2: Handled here since it's one of the more common operations and probably interesting to see in retrospect in case something goes wrong. Only messages where 'IsError' equals true are being logged. Normal messages are ignored by default, unless setting 'Magix.Brix.Components.Logger.LogAllMessages' equals true

[ActiveEvent(Name = "Magix.Core.Log- HARDLINK")]
protected void Magix_Core_Log(object sender, ActiveEventArgs e)

Level2: Will create one LogItem with the given 'LogItemType', 'Header', 'Message', 'ObjectID', 'ParentID', 'StackTrace' and so on, depending upon which data is actually being passed into it. Minimum requirement is 'Header'. If you append an Exclamation mark [!] as the last character of the LogItemType, the incident will be considered serious, and some sort of reaching out to the admin of the site might occur as a consequence, depending upon other states of the system. The Log action might often be overridden, if you need to force into this specific method for some reasons, you can add '-HARDLINK' to the action as you raise it

```
[ActiveEvent(Name = "Magix.Core.NewUserIDCookieCreated")]
protected void Magix_Core_NewUserIDCookieCreated(object sender, EventArgs e)
```

Level2: Handled since it's important information since it's highly likely it's a new visitor to the app/website

```
[ActiveEvent(Name = "Magix.QRCodes.CreateQRCode")]
protected void Magix_QRCodes_CreateQRCode(object sender, EventArgs e)
```

Level2: Logs the creation of every QR Code [and changes to them]

```
[ActiveEvent(Name = "Magix.Publishing.CreateChildWebPage")]
protected void Magix_Publishing_CreateChildWebPage(object sender, EventArgs e)
```

Level2: Handled to log every time somebody creates a WebPage

```
[ActiveEvent(Name = "Magix.Publishing.DeletePageObject- Confirmed")]
protected void Magix_Publishing_DeletePageObject_Confirmed(object sender, EventArgs e)
```

Level2: Logs every time a Web Page is being deleted

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
    EventArgs e)
```

Level2: Generic handler for handling some of the more common Active Types deletion scenarios

```
[ActiveEvent(Name = "Magix.Core.LoadActiveModule")]
protected void Magix_Core_LoadActiveModule(object sender, EventArgs e)
```

Level2: Every time a user is being warned, he will be shown a Message Box. We log all those message box warnings here, to document what the user has seen of warnings. We might also log other types of modules in this event handler

```
[ActiveEvent(Name = "Magix.Publishing.CreateUser")]
protected void Magix_Publishing_CreateUser(object sender, EventArgs e)
```

Level2: Logs the creation of Users

```
[ActiveEvent(Name = "Magix.PDF.CreatePDF- Book")]  
protected void Magix_PDF_CreatePDF_Book(object sender, EventArgs e)
```

Level2: Logs every time somebody tries to create a PDF book using Magix

```
[ActiveEvent(Name = "DBAdmin.Form.ViewClasses")]  
protected void DBAdmin_Form_ViewClasses(object sender, EventArgs e)
```

Level2: Logs the opening of the DB Manager

```
[ActiveEvent(Name = "Magix.FileExplorer.LaunchExplorer")]  
protected void Magix_FileExplorer_LaunchExplorer(object sender, EventArgs e)
```

Level2: Logs the opening of the File Manager

```
[ActiveEvent(Name = "Magix.Core.UserLoggedIn")]  
private void Magix_Core_UserLoggedIn(object sender, EventArgs e)
```

Level2: Handled here since it's one of the more common operations and probably interesting to see in retrospect in case something goes wrong

20. Games_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Logger.Games_Controller

Description:

Level2: Contains logic to load up games in menu etc

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]

protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Sink for allowing the System Games to become Menu Items in the Administrator Menu

[ActiveEvent(Name = "Magix.Games.OpenTetris")]

protected void Magix_Games_OpenTetris(object sender, EventArgs e)

Level2: Will open up Tetris and let the end user allow to play Tetris

21. DataBinding101_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaForms.DataBinding101_Controller

Description:

Level2: Will upon startup create an Image Button which 'flashes' which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Overridden to actually create our Form

22. MetaForms_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaForms.MetaForms_Controller

Description:

Level2: Contains logic to help build Meta Forms. Meta Forms are forms where you can almost in its entirety decide everything about how a View can be built

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Handled to create some default Meta Forms for the installation

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Will return the menu items needed to fire up 'View Meta Forms' forms for Administrator

[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender, EventArgs e)

Level2: Returns the MetaForm count back to dashboard

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]
protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, EventArgs e)

Level2: Will inject desktop icons for the Meta Form shortcut

```
[ActiveEvent(Name = "Magix.MetaForms.ViewForms")]
protected void Magix_MetaForms_ViewForms(object sender, EventArgs e)
```

Level2: Will show all Meta Forms within the system, and allow editing of them

```
[ActiveEvent(Name = "Magix.MetaForms.GetCopyMetaFormTemplateColumn")]
protected void Magix_MetaForms_GetCopyMetaFormTemplateColumn(object sender,
EventArgs e)
```

Level2: Will return a LinkButton, which once clicked will copy and edit a specific MetaForm

```
[ActiveEvent(Name = "Magix.MetaForms.CopyFormAndEdit")]
protected void Magix_MetaForms_CopyFormAndEdit(object sender, EventArgs e)
```

Level2: Performs a Deep-Copy of the MetaForm and start editing it immediately

```
[ActiveEvent(Name = "Magix.MetaForms.CopyForm")]
protected void Magix_MetaAction_CopyAction(object sender, EventArgs e)
```

Level2: Performs a Deep-Copy of the Form and returns the ID of the new MetaForm as 'NewID'

```
[ActiveEvent(Name = "Magix.MetaForms.CreateNewMetaForm")]
protected void Magix_MetaForms_CreateNewMetaForm(object sender, EventArgs e)
```

Level2: Creates a new MetaForm and edits it in content4

```
[ActiveEvent(Name = "Magix.MetaForms.EditForm")]
protected void Magix_MetaForms_EditForm(object sender, EventArgs e)
```

Level2: Will show the Edit Meta Forms form for editing a specific MetaForm

```
[ActiveEvent(Name = "Magix.MetaForms.GetControlTypes")]
protected void Magix_MetaForms_GetControlTypes(object sender, EventArgs e)
```

Level2: Will return the controls that the Meta Form builder has by default, such as Button, Label, CheckBox etc

```
[ActiveEvent(Name = "Magix.MetaForms.GetControlsForForm")]
protected void Magix_MetaForms_GetControlsForForm(object sender, EventArgs e)
```

Level2: Will return the Control tree hierarchy for the Meta Form

```
[ActiveEvent(Name = "Magix.MetaForms.EmptySingleWidgetInstance")]
protected void Magix_MetaForms_EmptySingleWidgetInstance(object sender, EventArgs e)
```

Level2: Expects a Widget in the 'Control' parameter, and the Type Node from the Meta Form as 'TypeNode'. Will empty and clear the control value of the widget

```
[ActiveEvent(Name = "Magix.MetaForms.SaveNodeSerializedFromMetaForm")]
protected void Magix_MetaForms_SaveNodeSerializedFromMetaForm(object sender,
    EventArgs e)
```

Level2: Will create MetaObject(s) from the given 'Object' node, with the given 'TypeName' if given, and save the Meta Objects to your data storage. If the 'Object' node contains an 'ID' child node, then the system will assume this is the ID for an existing Meta Object, fetch that Object and instead of creating a new object, update the values for the existing one. If it's updating an existing object, it will delete all existing properties of that specific object if 'CleanProperties' is true

```
[ActiveEvent(Name = "Magix.MetaForms.SerializeControlIntoNode")]
protected void Magix_MetaForms_SerializeControlIntoNode(object sender, EventArgs e)
```

Level2: Will serialize one Widget [BaseControl instance] into the given node. The control should exist as an instance in the 'Control' parameter, and the MetaForm serialized type node should exist in 'TypeNode'

```
[ActiveEvent(Name = "Magix.MetaForms.CreateControl")]
protected void Magix_MetaForms_CreateControl(object sender, EventArgs e)
```

Level2: Will create one of the default internally installed types for the Meta Form system, which includes e.g. Button, CheckBox and Label etc


```
[ActiveEvent(Name = "Magix.MetaForms.CreateRepeaterChildControlCollection")]
protected void Magix_MetaForms_CreateRepeaterChildControlCollection(object sender,
ActiveEventArgs e)
```

Level2: Will databind a Repeater towards the incoming Node structure according to how the Repeater is supposed to be DataBound, which again is according to its Info field. Will also Databind [obviously!] every Child Widget of your Repeater, and basically create as many rows as there are items in your Databinded Node.

```
[ActiveEvent(Name = "Magix.MetaForms.AppendControlToForm")]
protected void Magix_MetaForms_AppendControlToForm(object sender, ActiveEventArgs e)
```

Level2: Appends a new Control of type 'TypeName' to the given Meta Form 'ID' within its control with ID of 'ParentControl'. Let ParentControl be null or not defined to use the Root Form object of the Meta Form

```
[ActiveEvent(Name = "Magix.MetaForms.ChangeFormPropertyValue")]
protected void Magix_MetaForms_ChangeFormPropertyValue(object sender, ActiveEventArgs e)
```

Level2: Changes the 'ID' MetaForm's 'PropertyName' into 'Value' and saves the MetaForm property

```
[ActiveEvent(Name = "Magix.MetaForms.ShowAllActionsAssociatedWithFormWidgetEvent")]
protected void Magix_MetaForms_ShowAllActionsAssociatedWithFormWidgetEvent(object
sender, ActiveEventArgs e)
```

Level2: Will show the Actions associated with the Event 'EventName' on the MetaForm.Node with the ID of 'ID'

```
[ActiveEvent(Name = "Magix.MetaForms.ShowAllActionsAssociatedWithMainFormEvent")]
protected void Magix_MetaForms_ShowAllActionsAssociatedWithMainFormEvent(object sender,
ActiveEventArgs e)
```

Level2: Will show a Window with all the events for the Main Form of the given specific Meta Form 'ID' such that the user can add and remove items from the list

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the type information for editing Actions for form element on grid system

```
[ActiveEvent(Name = "Magix.MetaForms.GetPushActionUpTemplateColumn")]
protected void Magix_MetaForms_GetPushActionUpTemplateColumn(object sender,
ActiveEventArgs e)
```

Level2: Returns the Up column for the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.GetPushActionDownTemplateColumn")]
protected void Magix_MetaForms_GetPushActionDownTemplateColumn(object sender,
ActiveEventArgs e)
```

Level2: Returns the Down column for the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.PushActionUp")]
protected void Magix_MetaForms_PushActionUp_Down(object sender, ActiveEventArgs e)
```

Level2: Moves an Action up or down depending upon the event name

```
[ActiveEvent(Name = "Magix.MetaForms.ChangeSimplePropertyValue")]
protected void Magix_MetaForms_ChangeSimplePropertyValue(object sender, ActiveEventArgs
e)
```

Level2: Allows user to change name of Action Reference in MetaForms

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the list data for editing Actions for form element on grid system

```
[ActiveEvent(Name = "Magix.MetaForms.RemoveActionFromActionList")]
protected void Magix_MetaForms_RemoveActionFromActionList(object sender, ActiveEventArgs
e)
```

Level2: Sink for deleting Meta Form Action from Event

```
[ActiveEvent(Name = "Magix.MetaForms.RemoveActionFromActionList- Confirmed")]
protected void Magix_MetaForms_RemoveActionFromActionList_Confirmed(object sender,
ActiveEventArgs e)
```

Level2: Will actually remove the Action from the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.OpenAppendNewActionDialogue")]
protected void Magix_MetaForms_OpenAppendNewActionDialogue(object sender,
ActiveEventArgs e)
```

Level2: Will show the 'List of Actions form' for appending and editing and deleting actions associated with the specific Action on the specific Widget

```
[ActiveEvent(Name = "Magix.Forms.GetActionSelectActionTemplateColumn")]
protected void Magix_Forms_GetActionSelectActionTemplateColumn(object sender,
ActiveEventArgs e)
```

Level2: Returns the View Action Details Column control

```
[ActiveEvent(Name = "Magix.MetaForms.ShowActionDetails")]
protected void Magix_MetaForms_ShowActionDetails(object sender, ActiveEventArgs e)
```

Level2: Will open a Popup Window with the details about the Action, and the possibility of editing those details

```
[ActiveEvent(Name = "Magix.MetaForms.ConfigureFilterForColumns")]
protected void Magix_MetaForms_ConfigureFilterForColumns(object sender, ActiveEventArgs e)
```

Level2: Ads up Top=25 and nothing more before it forwards to 'DBAdmin.Form.ConfigureFilterForColumn'

```
[ActiveEvent(Name = "Magix.MetaForms.ActionWasSelected")]
protected void Magix_MetaForms_ActionWasSelected(object sender, ActiveEventArgs e)
```

Level2: Will append the given 'ID' Action into the given 'ParentPropertyName' Event name and save the MetaForm.Node

```
[ActiveEvent(Name = "Magix.MetaForms.MetaView_Form.GetTemplateColumnSelectForm")]
protected void Magix_MetaForms_MetaView_Form_GetTemplateColumnSelectForm(object sender, ActiveEventArgs e)
```

Level2: Returns the select list for selecting a Meta Form back to caller

```
[ActiveEvent(Name = "Magix.MetaForms.RaiseActionsFromActionString")]
protected void Magix_MetaForms_RaiseActionsFromActionString(object sender, EventArgs e)
```

Level2: Expects to get an action-list [named actions, separated by pipes []] which it will execute sequentially, in the order they are in. Expects to be raised from a WebPart, and needs 'ActionsToExecute' which is the piped list of Actions to execute

```
[ActiveEvent(Name = "Magix.MetaForms.AbsolutizeWidget")]
protected void Magix_MetaForms_AbsolutizeWidget(object sender, EventArgs e)
```

Level2: Will make sure the widget becomes absolutized and stored as such. Expects ID of MetaForm.Node through 'ID' and 'Left' and 'Top' pixel numbers defining where on the local coordinate system they're supposed to be displayed

```
[ActiveEvent(Name = "Magix.MetaForms.DeleteMetaFormWidgetFromForm")]
protected void Magix_MetaForms_DeleteMetaFormWidgetFromForm(object sender,
    EventArgs e)
```

Level2: Will delete the given 'ID' MetaForm.Node element from the MetaForm

```
[ActiveEvent(Name = "Magix.MetaForms.ViewCSharpCode")]
protected void Magix_MetaForms_ViewCSharpCode(object sender, EventArgs e)
```

Level2: Will show the creation code for creating this Meta Form using C# code

```
[ActiveEvent(Name = "Magix.MetaForms.OpenStyleBuilderForWidget")]
protected void Magix_MetaForms_OpenStyleBuilderForWidget(object sender, EventArgs e)
```

Level2: Will open up the Style Builder for creating Animations and Styles for your widget. The Event Handler needs an 'ID' parameter which defines which MetaForm.Node object to actually modify the style of upon saving of your style

```
[ActiveEvent(Name = "Magix.MetaForms.GetAnimations")]
protected void Magix_MetaForms_GetAnimations(object sender, EventArgs e)
```

Level2: Returns all existing Animations within 'Animations' as a list of 'Name'/'CSSClass' items

```
[ActiveEvent(Name = "Magix.Core.InitialLoading")]
protected void Magix_Core_InitialLoading(object sender, EventArgs e)
```

Level2: Includes the animation.css file onto every single first loading

```
[ActiveEvent(Name = "Magix.MetaForms.SetWidgetStyles")]
protected void Magix_MetaForms_SetWidgetStylesMagix_MetaForms_GetAnimations(object
sender, EventArgs e)
```

Level2: Updates and save all the incoming 'Style' values into the MetaForms.Node given through ID which should point to a Widget, with a 'Properties'/'Style' node

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
EventArgs e)
```

Level2: Overridden to make sure we can clean out our Content 4++ when deletion of MetaForms occurs

```
[ActiveEvent(Name = "Magix.MetaForms.LoadMetaForm")]
protected void Magix_MetaForms_LoadMetaForm(object sender, EventArgs e)
```

Level2: Expects a 'MetaFormName' pointing to a name of a specific Meta Form, and a 'Container' which normally would be between content1 to content7 or just child. The 'Container' must be based upon a WebPartTemplate which is of type MetaView_Form. Meaning, if there's not already a MetaForm in the location your 'Container' parameter is pointing to, then the Action will choke and break

```
[ActiveEvent(Name = "Magix.MetaForms.CopyWidget")]
protected void Magix_MetaForms_CopyWidget(object sender, EventArgs e)
```

Level2: Will create a copy of the widget [MetaForm.Node] with the given 'ID' which it will put into your 'clipboard', meaning you can paste in another copy of that widget later somewhere else into your MetaForms. Will copy the entire Widget and all of its Child Controls too

```
[ActiveEvent(Name = "Magix.MetaForms.PasteWidgetNodeIntoMetaForm")]
protected void Magix_MetaForms_PasteWidgetNodeIntoMetaForm(object sender,
EventArgs e)
```

Level2: Will clone and attach the newly cloned Node [given through 'PasteNode'] to the given MetaForm ['ID'] and if given as a child to the given 'ParentControl' Widget

```
[ActiveEvent(Name = "Magix.MetaAction.EditParam- O")]  
private void Magix_MetaAction_EditParam(object sender, ActiveEventArgs e)
```

Level2: Will initiate editing of Parameter for Action unless it's already being edited, at which point it'll be 'brought to front'

```
[ActiveEvent(Name = "Magix.MetaAction.EditParamReadOnly- O")]  
private void Magix_MetaAction_EditParamReadOnly(object sender, ActiveEventArgs e)
```

Level2: Will initiate viewing of Parameter for Action unless it's already being viewed, at which point it'll be 'brought to front'

23. CreateLoginForm_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaForms.CreateLoginForm_Controller

Description:

Level2: Will upon startup create a Login Form which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Overridden to actually create our Form

24. CreateHelloWorldForm_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaForms.CreateHelloWorldForm_Controller

Description:

Level2: Will upon startup create a Login Form which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Overridden to actually create our Form

25. CreateFlashingImageButton_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaForms.CreateFlashingImageButton_Controller

Description:

Level2: Will upon startup create an Image Button which 'flashes' which you can use as a 'starter kit' for your own logic, or directly consume as is, as a User Control

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Overridden to actually create our Form

26. MetaObject_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.MetaObject_Controller

Description:

Level2: Contains logic for editing, maintaining and viewing MetaObjects. MetaObjects are at the heart of the Meta Application System since they serve as the 'storage' for everything a view updates or creates through interaction with the end user

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]
protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, EventArgs e)

Level 2: Returns the Desktop Icon for launching Actions back to caller

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Returns menu event handlers for viewing MetaObjects

[ActiveEvent(Name = "Magix.MetaType.EditMetaObjects_UnFiltered")]
protected void Magix_MetaType_EditMetaObjects_UnFiltered(object sender, EventArgs e)

Level2: Will open up editing of the MetaObject directly, without any 'views' or other interferences. Mostly meant for administrators to edit objects in 'raw mode'

[ActiveEvent(Name = "Magix.MetaType.DeleteAll")]
protected void Magix_MetaType_DeleteAll(object sender, EventArgs e)

Level2: Will delete all actions with the given Criteria and refresh any grids

```
[ActiveEvent(Name = "DBAdmin.Grid.DeleteAllObjectsConfirmed- Override")]
protected void DBAdmin_Grid_DeleteAllObjectsConfirmed(object sender, EventArgs e)
```

Level2: Makes sure we're emptying our Container for any 'remnants' ...

```
[ActiveEvent(Name = "Magix.MetaType.GetCopyMetaObjectTemplateColumn")]
protected void Magix_MetaType_GetCopyMetaObjectTemplateColumn(object sender,
    EventArgs e)
```

Level3: Will return a 'Copy Template LinkButton' back to caller

```
[ActiveEvent(Name = "Magix.MetaType.CreateMetaObject")]
protected void Magix_MetaType_CreateMetaObject(object sender, EventArgs e)
```

Level2: Creates a new MetaObject with some default values and returns the ID of the new MetaObject as 'NewID'

```
[ActiveEvent(Name = "Magix.MetaType.CreateMetaObjectAndEdit")]
protected void Magix_MetaType_CreateMetaObjectAndEdit(object sender, EventArgs e)
```

Level2: Creates a new MetaObject with some default values, and lets the end user edit it immediately

```
[ActiveEvent(Name = "Magix.MetaType.EditONEMetaObject_UnFiltered")]
protected void Magix_MetaType_EditONEMetaObject_UnFiltered(object sender, EventArgs
    e)
```

Level2: Allows for editing the MetaObject directly without any Views filtering out anything

```
[ActiveEvent(Name = "Magix.MetaType.AppendChildMetaObjectToMetaObject")]
protected void Magix_MetaType_AppendChildMetaObjectToMetaObject(object sender,
    EventArgs e)
```

Level2: Will Append an existing MetaObject [ID] to another existing MetaObject [ParentID] as a child

```
[ActiveEvent(Name =  
"Magix.MetaType.AppendChildMetaObjectToMetaObjectAndEditParent")]  
protected void Magix_MetaType_AppendChildMetaObjectToMetaObjectAndEditParent(object  
sender, EventArgs e)
```

Level2: Will Append an existing MetaObject [ID] to another existing MetaObject [ParentID] as a child and immediately Edit the Parent MetaObject

```
[ActiveEvent(Name = "DBAdmin.Form.AppendObject- OverriddenForVisualReasons")]  
protected void DBAdmin_Form_AppendObject_OverriddenForVisualReasons(object sender,  
EventArgs e)
```

Level2: Calls DBAdmin.Form.AppendObject after overriding some 'visual properties' for the Grid system

```
[ActiveEvent(Name = "Magix.MetaType.CreateNewMetaObject- Value")]  
protected void Magix_MetaType_CreateNewMetaObject_Value(object sender, EventArgs e)
```

Level2: Creates a new 'Value Row' for our MetaObject [ID']. Returns the ID of the new Value object as 'NewID'

```
[ActiveEvent(Name = "Magix.MetaType.CreateNewMetaObject- Value- AndEdit")]  
protected void Magix_MetaType_CreateNewMetaObject_Value_AndEdit(object sender,  
EventArgs e)
```

Level2: Creates a new 'Value Row' for our MetaObject and Edits the MetaObject immediately

```
[ActiveEvent(Name = "Magix.MetaType.EditONEMetaObject_UnFiltered- ChildMetaObject")]  
protected void Magix_MetaType_EditONEMetaObject_UnFiltered_ChildMetaObject(object sender,  
EventArgs e)
```

Level2: Calls 'Magix.MetaType.EditONEMetaObject_UnFiltered' after changing the Container to display the module within. Allows editing of Child MetaObjects

```
[ActiveEvent(Name = "Magix.MetaType.GetMetaObjectChildrenTemplateColumn")]  
protected void Magix_MetaType_GetMetaObjectChildrenTemplateColumn(object sender,  
EventArgs e)
```

Level3: Returns a LinkButton with no Children back to caller upon which clicked will start editing the Children collection of objects within the MetaObject

```
[ActiveEvent(Name = "Magix.Publishing.RemoveChildObject")]  
protected void Magix_Publishing_RemoveChildObject(object sender, EventArgs e)
```

Level2: Will remove the Child MetaObject ['ID'] from the Parent MetaObject ['ParentID'] collection of children. Notice the child object will NOT be deleted, only 'unreferenced out of' the parent MetaObject

```
[ActiveEvent(Name = "Magix.Publishing.RemoveChildObjectAndEdit")]  
protected void Magix_Publishing_RemoveChildObjectAndEdit(object sender, EventArgs e)
```

Level2: Will remove the Child MetaObject ['ID'] from the Parent MetaObject ['ParentID'] collection of children. Notice the child object will NOT be deleted, only 'unreferenced out of' the parent MetaObject. Will instantly edit the Parent MetaObject.

```
[ActiveEvent(Name = "Magix.MetaType.GetMetaObjectValuesTemplateColumn")]  
protected void Magix_MetaType_GetMetaObjectValuesTemplateColumn(object sender,  
EventArgs e)
```

Level3: Will return a TextAreaEdit, from which the Value of the Value object belonging to the MetaObject can be edited, and a LinkButton, from which the entire object can be deleted, back to caller

```
[ActiveEvent(Name = "Magix.MetaType.GetMetaObjectValuesNAMETemplateColumn")]  
protected void Magix_MetaType_GetMetaObjectValuesNAMETemplateColumn(object sender,  
EventArgs e)
```

Level3: Will return an InPlaceEdit back to caller, since having Carriage Returns in a Property Name would only serve to be ridiculous

```
[ActiveEvent(Name = "Magix.MetaType.DeleteMetaObject")]  
protected void Magix_MetaType_DeleteMetaObject(object sender, EventArgs e)
```

Level2: Will ask the user for confirmation to assure he really wants to delete the specific MetaObject ['ID'], and if the user confirms will delete that object

```
[ActiveEvent(Name = "Magix.MetaType.DeleteObjectRaw- Confirmed")]  
protected void Magix_MetaType_DeleteObjectRaw_Confirmed(object sender, EventArgs e)
```

Level2: Implementation of deletion of MetaObject after user has confirmed he really wants to delete it

```
[ActiveEvent(Name = "DBAdmin.Common.CreateObjectAsChild")]
protected void DBAdmin_Common_CreateObjectAsChild(object sender, ActiveEventArgs e)
```

Level2: Here only to make sure Grids are updated if we're adding a child MetaObject to another MetaObject

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Clears from content4 and out

```
[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Returns the number of MetaObjects in the system back to caller and the name of the Event needed to show all MetaObjects in the system

```
[ActiveEvent(Name = "Magix.MetaType.SetMetaObjectValue")]
protected void Magix_MetaType_SetMetaObjectValue(object sender, ActiveEventArgs e)
```

Level2: Will either update the existing Value or create a new Value with the given 'Name' and make sure exists within the MetaObject ['ID']

```
[ActiveEvent(Name = "DBAdmin.DynamicType.ChangeValue")]
protected void DBAdmin_DynamicType_ChangeValue(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure we can traverse our MetaObjects in META mode [front-web, showing grids and views of Meta Objects]

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObject")]
protected void DBAdmin_DynamicType_GetObject(object sender, ActiveEventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be seen 'front-web'

```
[ActiveEvent(Name = "DBAdmin.Data.GetObject")]
protected void DBAdmin_Data_GetObject(object sender, ActiveEventArgs e)
```

Level2: For flattening out our Meta Object so that properties shows as values

```
[ActiveEvent(Name = "Magix.Meta.EditMetaObject")]  
protected void Magix_Meta_EditMetaObject(object sender, EventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be edited 'front-web'

```
[ActiveEvent(Name = "Magix.Meta.ChangeAllPropertyValues")]  
protected void Magix_Meta_ChangeAllPropertyValues(object sender, EventArgs e)
```

Level2: Will change the value of every single property 'PropertyName' object of the given 'TypeName' to the given 'NewValue'

```
[ActiveEvent(Name = "Magix.Meta.CreateNewMetaObject")]  
protected void Magix_Meta_CreateNewMetaObject(object sender, EventArgs e)
```

Level2: Will take the given 'Properties' and save as properties for a new Meta Object with the typename of 'TypeName' and Parent Meta object being 'ParentID', if not null

```
[ActiveEvent(Name = "Magix.Meta.DeleteMetaObject")]  
protected void Magix_Meta_DeleteMetaObject(object sender, EventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be deleted 'front-web'

```
[ActiveEvent(Name = "Magix.Meta.DeleteMetaObject- Confirmed")]  
protected void Magix_Meta_DeleteMetaObject_Confirmed(object sender, EventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can be seen 'front-web'. Confirmation, actual deletion

```
[ActiveEvent(Name = "Magix.Meta.ChangeMetaObjectValue")]  
protected void Magix_Meta_ChangeMetaObjectValue(object sender, EventArgs e)
```

Level2: Handled to make sure "META mode" MetaObjects can have their values changed 'front-web'

```
[ActiveEvent(Name = "Magix.MetaType.CopyMetaObject")]  
private void Magix_MetaType_CopyMetaObject(object sender, EventArgs e)
```

Level2: Will copy the incoming MetaObject ['ID']

27. UploaderPlugin_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.UploaderPlugin_Controller

Description:

Level2: Contains logic for being able to upload files to the server

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.FileUploaded")]

protected void Magix_Core_FileUploaded(object sender, EventArgs e)

Level2: Will save the given 'File' to the given 'Folder' with the given 'FileName' and then raise 'ActionName', unless it's either null or empty, or it's "NO-ACTION". If the Action starts with 'finished:', the action will only be raised when an entire batch is done. Otherwise the action will be raise consecutively for every file within the same batch

[ActiveEvent(Name = "Magix.Core.FileBatchUploadFinished")]

protected void Magix_Core_FileBatchUploadFinished(object sender, EventArgs e)

Raised when an entire batch of files are done. If the 'ActionName' starts with 'finished:', the Action given in the | separated string will be consecutively raised

28. MetaAction_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.MetaAction_Controller

Description:

Level2: Contains logic for Actions which are wrappers around ActiveEvents for the end user to be able to raise his own events, with his own data in the Node structure. The MetaAction system is at the core of the Meta Application system wince without it the end user cannot create his own types of events or Actions

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Will initialize all the Actions which are Overriding existing Active Events

[ActiveEvent(Name = "Magix.MetaAction.ReInitializeOverriddenActions")]
protected void Magix_Meta_ReInitializeOverriddenActions(object sender, ActiveEventArgs e)

Level2: Will Reinitialize all the Overridden Actions

[ActiveEvent(Name = "Magix.MetaAction.ClearOverriddenActions")]
protected void Magix_Meta_ClearOverriddenActions(object sender, ActiveEventArgs e)

Level2: Will Clear all the Overridden Actions

[ActiveEvent(Name = "Magix.MetaAction.RaiseOverriddenAction")]
protected void Magix_Meta_RaiseOverriddenAction(object sender, ActiveEventArgs e)

Level2: Helper to make sure our Overriding Actions are being raised

```
[ActiveEvent(Name = "Magix.MetaAction.RaiseOverriddenActionCallBase")]
protected void Magix_Meta_RaiseOverriddenActionCallBase(object sender, EventArgs e)
```

Level2: Helper to make sure our Overriding Actions are being raised

```
[ActiveEvent(Name = "Magix.MetaAction.RaiseOverriddenActionCallBaseFirst")]
protected void Magix_Meta_RaiseOverriddenActionCallBaseFirst(object sender, EventArgs e)
```

Level2: Helper to make sure our Overriding Actions are being raised

```
[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]
protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, EventArgs e)
```

Level 2: Returns the Desktop Icon for launching Actions back to caller

```
[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)
```

Level2: Will return the menu items needed to fire up 'View Meta Actions' forms for Administrator

```
[ActiveEvent(Name = "Magix.MetaType.ViewActions")]
protected void Magix_MetaType_ViewActions(object sender, EventArgs e)
```

Level2: Will show a Grid containing all Meta Actions within the system

```
[ActiveEvent(Name = "Magix.MetaActions.SearchActions")]
protected void Magix_MetaActions_SearchActions(object sender, EventArgs e)
```

Level2: Will show a 'search for Action and select' type of Grid to the end user so that he can select and append an action into whatever collection wants to contain one

```
[ActiveEvent(Name = "Magix.MetaAction.ActionWasSelected")]
protected void Magix_MetaAction_ActionWasSelected(object sender, EventArgs e)
```

Level2: Raise 'ParentPropertyName', first setting the ActionName to the Action found in 'ID'. Helper method for editing Actions in Views

```
[ActiveEvent(Name = "DBAdmin.Data.GetContentsOfClass- Filter-Override")]
protected void DBAdmin_Data_GetContentsOfClass_Filter_Override(object sender,
ActiveEventArgs e)
```

Level2: Basically just overrides 'DBAdmin.Data.GetContentsOfClass' to allow for adding some custom Criterias for our Search Box. Puts 'Filter' into a Like expression on the Name column before calling 'base class'

```
[ActiveEvent(Name = "Magix.MetaAction.GetCopyActionTemplateColumn")]
protected void Magix_MetaAction_GetCopyActionTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Returns a LinkButton that will allow for Deep-Copying the selected Action

```
[ActiveEvent(Name = "Magix.MetaAction.CopyAction")]
protected void Magix_MetaAction_CopyAction(object sender, ActiveEventArgs e)
```

Level2: Performs a Deep-Copy of the Action and returns the ID of the new Action as 'NewID'

```
[ActiveEvent(Name = "Magix.MetaAction.CopyActionAndEdit")]
protected void Magix_MetaAction_CopyActionAndEdit(object sender, ActiveEventArgs e)
```

Level2: Performs a Deep-Copy of the Action and start editing the Action immediately

```
[ActiveEvent(Name = "Magix.Meta.CreateAction")]
protected void Magix_Meta_CreateAction(object sender, ActiveEventArgs e)
```

Level2: Creates a new Default Action and returns the ID of the new Action as 'NewID'

```
[ActiveEvent(Name = "Magix.Meta.CreateActionAndEdit")]
protected void Magix_Meta_CreateActionAndEdit(object sender, ActiveEventArgs e)
```

Level2: Creates a new Default Action and starts editing it immediately

```
[ActiveEvent(Name = "Magix.Meta.EditAction")]
protected void Magix_Meta_EditAction(object sender, ActiveEventArgs e)
```

Level2: Edits the given Action ['ID'], with all its properties, parameters and so on. Also creates a 'Run' button which the end user can click to run the action

```
[ActiveEvent(Name = "Magix.MetaAction.FindOverrideActiveEvent")]
protected void Magix_MetaAction_FindOverrideActiveEvent(object sender, ActiveEventArgs e)
```

Level2: Will show a Search Modal Window which will allow you to find any Active Event handler within the system

```
[ActiveEvent(Name = "Magix.Meta.SelectActiveEvent")]
protected void Magix_Meta_SelectActiveEvent(object sender, ActiveEventArgs e)
```

Level2: Sink for selecting an Active Event to encapsulate within an Action

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode- AT")]
protected void DBAdmin_DynamicType_GetObjectsNode_AT(object sender, ActiveEventArgs e)
```

Level2: Sink for finding all Active Events within the system Application Pool

```
[ActiveEvent(Name = "Magix.MetaAction.RaiseAction- FromActionScreen")]
protected void Magix_MetaAction_RaiseAction_FromActionScreen(object sender,
ActiveEventArgs e)
```

Level2: Will forward to RaiseAction, and show a Message to user ...

```
[ActiveEvent(Name = "Magix.Meta.DeleteParameter")]
protected void Magix_Meta_DeleteParameter(object sender, ActiveEventArgs e)
```

Level2: Deletes the Action.ActionParams given ['ID'] and updates a lot of UI properties. Raises 'Magix.Core.GetSelectedTreeItem' to get the ID of which ActionParams to actually delete

```
[ActiveEvent(Name = "Magix.Meta.CreateParameter")]
protected void Magix_Meta_CreateParameter(object sender, ActiveEventArgs e)
```

Level2: Creates a new Parameter and attaches to either the Selected Parameter, or if none selected, the Action directly on root level. Depends upon the 'Magix.Core.GetSelectedTreeItem' event to get to know whether or not it should add the action to a specific ActionParam as a Child or directly upon the root level of the Action given through 'ID'

```
[ActiveEvent(Name = "Magix.MetaAction.DeleteMetaAction")]  
protected void Magix_MetaAction_DeleteMetaAction(object sender, EventArgs e)
```

Level2: Will ask the user for confirmation about deleting the given Action ['ID'], and if the end user confirms, the Action will be deleted

```
[ActiveEvent(Name = "Magix.MetaAction.DeleteMetaAction- Confirmed")]  
protected void Magix_MetaAction_DeleteMetaAction_Confirmed(object sender, EventArgs e)
```

Level2: Will call 'DBAdmin.Common.ComplexInstanceDeletedConfirmed' which again [hopefully] will delete the given Action

```
[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]  
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender, EventArgs e)
```

Level2: Returns menu items for dashboard functionality to be able to click and view Actions from Dashboard

```
[ActiveEvent(Name = "Magix.Core.EventClickedWhileDebugging")]  
protected void Magix_Core_EventClickedWhileDebugging(object sender, EventArgs e)
```

Level2: Will take an incoming 'EventName' with an optionally attached 'EventNode' structure and create an Action out of it

```
[ActiveEvent(Name = "Magix.MetaAction.RaiseAction")]  
protected void Magix_Meta_RaiseEvent(object sender, EventArgs e)
```

Level2: Will take an incoming Action ['ActionID' OR 'ActionName'] and run it. Will merge the incoming parameters with the Params of the Action, giving the 'incoming Parameters' preference over the Params associated with Action

```
[ActiveEvent(Name = "Magix.MetaForms.RemoveActionFromActionList3")]  
protected void Magix_MetaForms_RemoveActionFromActionList3(object sender, EventArgs e)
```

Level2: Sink for deleting Meta Form Action from Event

```
[ActiveEvent(Name = "Magix.MetaForms.RemoveActionFromActionList3- Confirmed")]
protected void Magix_MetaForms_RemoveActionFromActionList3_Confirmed(object sender,
ActiveEventArgs e)
```

Level2: Will actually remove the Action from the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.OpenAppendNewActionDialogue3")]
protected void Magix_MetaForms_OpenAppendNewActionDialogue3(object sender,
ActiveEventArgs e)
```

Level2: Will show the 'List of Actions form' for appending and editing and deleting actions associated with the specific Action on the specific Widget

```
[ActiveEvent(Name = "Magix.MetaForms.ActionWasSelected3")]
protected void Magix_MetaForms_ActionWasSelected3(object sender, ActiveEventArgs e)
```

Level2: Will append the given 'ID' Action into the given 'ParentPropertyName' Event name and save the MetaForm.Node

```
[ActiveEvent(Name = "Magix.MetaActions.ChangeSimplePropertyValue3")]
protected void Magix_MetaActions_ChangeSimplePropertyValue3(object sender, ActiveEventArgs e)
```

Level2: Allows user to change name of Action Reference in WebParts

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the type information for editing Actions for form element on grid system

```
[ActiveEvent(Name = "Magix.MetaForms.GetPushActionUpTemplateColumn3")]
protected void Magix_MetaForms_GetPushActionUpTemplateColumn3(object sender,
ActiveEventArgs e)
```

Level2: Returns the Up column for the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.GetPushActionDownTemplateColumn3")]
protected void Magix_MetaForms_GetPushActionDownTemplateColumn3(object sender,
ActiveEventArgs e)
```

Level2: Returns the Down column for the Action list

```
[ActiveEvent(Name = "Magix.MetaForms.PushActionUp3_Down3")]
protected void Magix_MetaForms_PushActionUp3_Down3(object sender, ActiveEventArgs e)
```

Level2: Moves an Action up or down depending upon the event name

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the list data for editing Actions for WebPart element

```
[ActiveEvent(Name = "Magix.MetaAction.EditParam")]
private void Magix_MetaAction_EditParam(object sender, ActiveEventArgs e)
```

Level2: Will initiate editing of Parameter for Action unless it's already being edited, at which point it'll be 'brought to front'

```
[ActiveEvent(Name = "Magix.MetaAction.EditParamReadOnly")]
private void Magix_MetaAction_EditParamReadOnly(object sender, ActiveEventArgs e)
```

Level2: Will initiate viewing of Parameter for Action unless it's already being viewed, at which point it'll be 'brought to front'

```
[ActiveEvent(Name =
"Magix.MetaAction.GetMetaActionParameterTypeNameTemplateColumn")]
private void Magix_MetaAction_GetMetaActionParameterTypeNameTemplateColumn(object
sender, ActiveEventArgs e)
```

Level3: Returns a SelectList with the opportunity for the end user to select which type [system-type, native] the specific parameter should be converted to before Action is being ran

```
[ActiveEvent(Name = "Magix.MetaAction.GetActionItemTree")]
private void Magix_Meta_GetActionItemTree(object sender, ActiveEventArgs e)
```

Level2: Returns a tree structure containing all the Action's Parameters to the caller

```
[ActiveEvent(Name = "Magix.Publishing.GetSelectActionForModuleControl")]  
private void Magix_Publishing_GetSelectActionForModuleControl(object sender, EventArgs  
e)
```

Level2: Sink for selecting Actions for Modules

29. CommonActions_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.CommonActions_Controller

Description:

Level2: Contains common end user useful actions which doesn't really belong any particular place, but which can still be immensely useful for scripting purposes. Perceive these as pugins, or extra candy if you wish. Often they're simplifications of other more hard core Active Events. This is one of the most interesting classes for you to look at in order to figure out what events to start using

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Common.GetUserSetting")]

protected void Magix_Common_GetUserSetting(object sender, EventArgs e)

Level2: Returns the 'Name' setting for the current logged in User. Sets the setting to 'Default' if no existing value is found. Both are mandatory params. Returns new value as 'Value'

[ActiveEvent(Name = "Magix.Common.SetUserSetting")]

protected void Magix_Common_SetUserSetting(object sender, EventArgs e)

Level2: Sets the given setting 'Name' to the given value 'Value' for the currently logged in User

[ActiveEvent(Name = "Magix.Common.AddToCache")]

protected void Magix_Common_AddToCache(object sender, EventArgs e)

Level2: Will add the given 'Value' with the given 'Key' to the Caching.Cache Page object. If the Value of 'Value' contains anything but null or " it will be serialized as JSON, otherwise the Value object of the given node will be serialized

```
[ActiveEvent(Name = "Magix.Common.GetFromCache")]  
protected void Magix_Common_GetFromCache(object sender, EventArgs e)
```

Level2: Will return either the 'Key' which must be exact match, or the first item of 'Contains' that contains the given string as its key

```
[ActiveEvent(Name = "Magix.Common.RemoveFromCache")]  
protected void Magix_Common_RemoveFromCache(object sender, EventArgs e)
```

Level2: Will remove the given 'Key' item from your Caching. Cache object associated with the Page object. If 'Key' is a direct hit, it will only remove one entry, if not, it will remove everything containing 'Key' from its cache

```
[ActiveEvent(Name = "Magix.Common.ClearCache")]  
protected void Magix_Common_ClearCache(object sender, EventArgs e)
```

Level2: Clears the Caching.Cache associated with the Page object

```
[ActiveEvent(Name = "Magix.System.execute")]  
protected void Magix_System_execute(object sender, EventArgs e)
```

Level2: Executes the given node recursively. This is the Magix Turing Executor which allows for logic to be implemented without having to code. The Magix Turing Executor can modify code at its instruction pointer if you wish. You can create selfmodifiable code which changes the code as it is being executed almost. Make sure you understand Expressions to get the most out of the Magix Turing Executor.

The Magix Turing Execution engine allows for these keywords; 'if', 'else if', 'else', 'execute', 'raise', 'throw', 'foreach', 'set', 'untie' and 'while'. 'if' will evaluate its value as either a 1 or 3 compund expression statement. 'if' might look like this: {root[Object][Name].Value} == John Doe. The previous would evaluate to true if the specific node contained John Doe as its value. 'else if' is an if, but will only be evaluated if none of the previous 'if' or 'else if' did not evaluate to true. 'else' is what will occur if none of the previous 'if' and 'else if' did not evaluate to true. 'execute' creates a new scope, where you can comment as a part of the Value of the 'execute' node. 'execute' will automatically untie itself after it's execute to avoid redundant logic in passing around data and such. 'raise' will raise the Active Event from the Value of the 'raise' node and pass in the 'raise' node itself as the parameters to your Active Event. You can also give 'raise' an Expression which must return a Node List, at which point that Expression's Node will be passed into the Active Event instead. Hint; By combining 'raise' with Action Overrides, you can raise Actions and concatenate Actions this way according to the flow of your business logic. 'throw' will raise an ApplicationException with the given error message from the Value of the 'throw' node. 'foreach' will iterate through every single node returned from the Node expression and run through its own children nodes as an execution block, once for every node returned from the Node expression, and pass in 'idx' as the iterator or this node from the iteration process. 'set' will set the Name, Value or Children of an Expression as its first parameter to the value of its second parameter. First parameter must be an Expression, while second parameter can either be an expression or a fixed value. Parameters in 'set' are separated by '='. If the left hand expression is a node list, then the right hand expression also must be a node list. 'untie' will untie its parent Node regardless of what it is. Useful for appending at the end of execute statements to remove them after the logic is executed, or clean up other types of data. 'untie' can take an optional expression as its Value, which should return a Node list if defined, whos parent will be untied. If no Expression is given, 'untie' will untie itself and execute its child nodes. No execution of childnodes will occur unless an Expression is given. 'when' will execute as long as its expression is true. The 'when' expression will be evaluated exactly the same is an 'if' expression, except the Expression will be evaluated first before entering the loop, and only entering if evaluated to true. If entered, it will execute the entire loop, re-evaluate the same Expression unless it's changed somehow, and only loop back to the beginning of the loop if the Expression still evaluates to true

```
[ActiveEvent(Name = "Magix.System.set")]
protected void Magix_System_set(object sender, ActiveEventArgs e)
```

Level2: Expects two values, the first parameter must must be an expression, the second parameter might either be an expression or a fixed static value. The two parameters are separated by '=' and expected to be found in the Value of the current 'set' node. Warning, this will create the left hand parameter node path if not existing, and it will also set the value of that path to null of no right hand side value is found. The left hand side of your component expression can be either '.Value', '.Name' or a resulting Node list. If the left hand side is a Node list Expression, then all previous nodes in the Expression result, if any, will be removed, and all nodes from the right hand side result will be cloned [deep copy] and put into the left hand side's Node list

```
[ActiveEvent(Name = "Magix.System.if")]  
protected void Magix_System_if(object sender, EventArgs e)
```

Level2: Branching according to statement contained within 'if' Value node. 'if' can take either one or three components. If one components is given, the if will evaluate to true if the components contains anything but "" and null. If 3 components are given, it will evaluate to true depending upon the comparison operator that must be the second components. Legal operators are '>', '<', '==', '!=', '>=' and '<='. The quotes are not a parts of the components, and only here for illustrative purposes. All comparisons are done as if both sides were strings, meaning 4 will be evaluated more than 33. So numbers cannot be correctly compared without additional logic on your behalf for instance. 'if' will if it evaluates to true work like an 'execute' scope, except it will not untie itself in any ways after execution

```
[ActiveEvent(Name = "Magix.System.else if")]  
protected void Magix_System_else_if(object sender, EventArgs e)
```

Level2: Branching according to statement contained within 'else if' Value node. 'else if' can take either one or three components. If one components is given, the else if will evaluate to true if the components contains anything but "" and null. If 3 components are given, it will evaluate to true depending upon the comparison operator that must be the second components. Legal operators are '>', '<', '==', '!=', '>=' and '<='. The quotes are not a parts of the components, and only here for illustrative purposes. All comparisons are done as if both sides were strings, meaning 4 will be evaluated more than 33. So numbers cannot be correctly compared without additional logic on your behalf for instance. 'else if' must directly follow either another 'else if' or an 'if'. the 'else if' will only be tried if none of the previous 'if' and 'else if' did not evaluate to true. 'else if' will if it evaluates to true work like an 'execute' scope, except it will not untie itself in any ways after execution

```
[ActiveEvent(Name = "Magix.System.else")]  
protected void Magix_System_else(object sender, EventArgs e)
```

Level2: Branching according to statement contained within previous 'if' or 'else if'. The 'else' will only be evaluated if none of the previous 'else if' or 'if' evaluated to true. 'else' will if it is being executed work like an 'execute' scope, except it will not untie itself in any ways after execution

```
[ActiveEvent(Name = "Magix.System.raise")]  
protected void Magix_System_raise(object sender, ActiveEventArgs e)
```

Level2: Raises the Active Event defined in the Value of the node. If you concatenate an Expression inside of normal parentheses, e.g.; 'Magix.Core.ShowMessage({root[MessageNode]})', then the given value Node list of your Expression, inside of your parentheses, will be passed in instead of the default which is the 'raise' node itself

```
[ActiveEvent(Name = "Magix.System.throw")]  
protected void Magix_System_throw(object sender, ActiveEventArgs e)
```

Level2: Raises an ApplicationException with the error message of the exception being the Value of the node. Will instantly stop execution of everything, unwind the stack, probably log some error message, and return control back to the caller

```
[ActiveEvent(Name = "Magix.System.foreach")]  
protected void Magix_System_foreach(object sender, ActiveEventArgs e)
```

Level2: Loops through all nodes in Expression and raises 'execute' with every one as a 'this' instance. 'foreach' needs an Expression, which must return a Node list, in its Value part. This node list will be iterated upon, and the content of the 'foreach' will function as an execute block, though without any untie, once for every single node in your result set of the Expression within the 'foreach' Value. The current iterator object of your foreach iteration can be de-referenced using 'idx', e.g. '{idx[Name].Value}'

```
[ActiveEvent(Name = "Magix.System.while")]  
protected void Magix_System_while(object sender, ActiveEventArgs e)
```

Level2: Branching according to statement contained within 'while' Value node. 'while' can take either one or three components. If one components is given, the while will evaluate to true if the components contains anything but "" and null. If 3 components are given, it will evaluate to true depending upon the comparison operator that must be the second components. Legal operators are '>', '<', '==', '!=', '>=' and '<='. The quotes are not a parts of the components, and only here for illustrative purposes. All comparisons are done as if both sides were strings, meaning 4 will be evaluated more than 33. So numbers cannot be correctly compared without additional logic on your behalf for instance. 'while' will if it evaluates to true work like an 'execute' scope, except it will not untie itself in any ways after execution. When it is done executing, it will re-evaluate its Value Expression and start all over again if the Expression still evaluates to true

```
[ActiveEvent(Name = "Magix.System.untie")]
protected void Magix_System_untie(object sender, EventArgs e)
```

Level2: Will untie the current Node. Can also be given an Expression that returns a Node list, at which it will untie that node. Afterwards, all child nodes will become executed, but only if no Expression is given. Which creates a useful way to start execution paths, on their own temporary data, which does not mess up the DataSource node structure of your Module, etc

```
[ActiveEvent(Name = "Magix.Common.ToggleCSSClassOfWidgetOrControl")]
protected void Magix_Common_ToggleCSSClassOfWidgetOrControl(object sender,
    EventArgs e)
```

Level2: Changes the CSS class of the Widget or Server Side Control with the given 'ID'. The Control must be of type BaseWebControl. The CSS class will be rolled according to all values in 'Values' parameter

```
[ActiveEvent(Name = "Magix.MetaTypes.RollThroughEnumValuesForProperty")]
protected void Magix_MetaTypes_RollThroughEnumValuesForProperty(object sender,
    EventArgs e)
```

Level2: Will roll a property value through the different values within 'Enums' node

```
[ActiveEvent(Name = "Magix.Common.GetActiveTypeObjects")]
protected void Magix_Common_GetActiveTypeObjects(object sender, EventArgs e)
```

Level2: Will return objects of given 'TypeName' back to caller as 'Objects' with every instance within 'Objects' will contain one 'ID' node, and another node called 'Properties' which will contain values for every Property within that object within a flat hierarchy

```
[ActiveEvent(Name = "Magix.MetaView.ScrollClientToTop")]
protected void Magix_MetaView_ScrollClientToTop(object sender, EventArgs e)
```

Level2: Will scroll the Client browser all the way back to the top over a period of 500 Milliseconds [.5 seconds]

```
[ActiveEvent(Name = "Magix.Common.PutGETParameterIntoDataSource")]
protected void Magix_Common_PutGETParameterIntoDataSource(object sender,
    EventArgs e)
```

Level2: Will put the given 'ParamName' GET parameter into the given node under the name of the Parameter

[ActiveEvent(Name = "Magix.Common.GetMetaObjectGraph")]
protected void Magix_Common_GetMetaObjectGraph(object sender, EventArgs e)

Level2: Will return an entire Graph of an object as 'Object', meaning all its child objects too. Expects to find the ID of the MetaObject to fetch in the 'ID' parameter

[ActiveEvent(Name = "Magix.MetaObjects.GetMetaObjects")]
protected void Magix_MetaObjects_GetMetaObjects(object sender, EventArgs e)

Level2: Will return from 'Start' to 'End', sorted according to newest first Meta Objects of type 'MetaTypeName' in the 'Objects' parameter. If you set 'Ascending' to true, it will sort according to oldest first. If you give no 'End' parameter, then it will fetch the 10 items starting from start, if you give no 'Start' parameter, it'll fetch from the beginning

[ActiveEvent(Name = "Magix.MetaObjects.GetNextMetaObjects")]
protected void Magix_MetaObjects_GetNextMetaObjects(object sender, EventArgs e)

Level2: Requires 'Start', 'End' and 'MetaTypeName', just like its GetMetaObjects event counterpart. Will first UnTie the 'Objects' node, then it will increase 'Start' and 'End' by 'End' - 'Start', and the same increase for 'End', and then return these objects to the caller in 'Objects'. You can also add 'Ascending' as a boolean to signify sorting order. It will also return its new 'Start' and 'End' value according to which objects were actually fetched. Will display a message to user if there are no more objects to be fetched

[ActiveEvent(Name = "Magix.MetaObjects.GetPreviousMetaObjects")]
protected void Magix_MetaObjects_GetPreviousMetaObjects(object sender, EventArgs e)

Level2: Requires 'Start', 'End' and 'MetaTypeName', just like its GetMetaObjects event counterpart. Will first UnTie the 'Objects' node, then it will decrease 'Start' and 'End' by 'End' - 'Start', and the same decrease for 'End', and then return these objects to the caller in 'Objects'. You can also add 'Ascending' as a boolean to signify sorting order. It will also return its new 'Start' and 'End' value according to which objects were actually fetched. Will display a message to user if there are no more objects to be fetched

```
[ActiveEvent(Name = "Magix.Common.ValidateObjectProperty")]  
protected void Magix_Common_ValidateObjectProperty(object sender, ActiveEventArgs e)
```

Level2: Will validate the given object, sent from a MetaView SingleView, such that it conforms in its 'PropertyName' value towards the 'Type' parameter, which can either be 'email' or 'number'. Meaning if the MetaView has a Property called 'Email', then if the property was written such; 'email:Email' - Then if the user writes anything but a legal Email address, according to RFC 5321, minus length restrictions, in the 'Email' field, and exception will be thrown, and the current execution chain of Actions will be stopped and no more Actions will be executed. The Message of the Exception will be, at least to some extent, friendly. For implementers of plugins, handle this Active Event, and check towards the 'Type' parameter which then will contain your chosen prefix. Prefixes for this method are 'email', 'number', 'mandatory', 'url' and 'full-name'. PS! Some things might get changed during the course of this event, for instance if you have a 'url' field, without http:// or https:// in the beginning, those characters will be appended. 'full-name' will normalize the given value such that it says 'Hansen, Thomas', it will also Auto Capitalize. If name is given like thomas hansen it will assume first name is first. If there's a comma, it will assert that the last name(s) are behind the comma, and the first name(s) before it. It will still produce the result 'Hansen, Thomas Polterguy Hooeog' from e.g. 'thomas poLTerguy Hooeog hanseN', or from 'hansen, thomas POLTERGUY hooeog'. While 'Hooeog Hansen, Thomas Polterguy' will emerge from e.g. 'hooeog hansen, thomas polterGUY'. Unless the 'AcceptNull' is given, most controls will not accept an empty value. While for 'full-name', unless 'AcceptHalfName' is given, or 'AcceptNull' is given, it will either accept only a full name, meaning at least two names or only one name if 'AcceptHalfName' is true. While unless 'AcceptNull' is null, it will still need at least one name, with more than 2 letters within. It will only accept one comma. If the Parameter 'FirstFirst' equals to true, it'll put the First Name BEFORE the Last Name, and use no commas

```
[ActiveEvent(Name = "Magix.Common.SendEmail")]  
protected void Magix_Common_SendEmail(object sender, ActiveEventArgs e)
```

Level2: Simplification of 'Magix.Core.SendEmail', will among other things using the User.Current as sender unless explicitly overridden. Will also unless to email address is given, send email to yourself. Takes these parameters 'Header', 'Body', 'Email' [from email], 'From' [name] 'To' [which can be a list of emails or one email]


```
[ActiveEvent(Name = "Magix.Common.Transform")]  
protected void Magix_Common_TransformNode(object sender, ActiveEventArgs e)
```

Level2: Will transform the input node according to the expressions within its 'Expression' parameter node and its children. Will return the transformed Node as its 'Expression' return value, which will become a completely new node, depending upon the expressions transforming it. Every node containing an expression as either its Name or Value, will become transformed into the result of its expression. All other nodes will be considered static values and left untouched. Only nodes within the 'Expression' parameter will be transformed. If there are no 'Expression' node, then the entire hierarchy will be treated as an Expression. Regardless, the result will be returned in the 'Expression' node, which will become the result. Expressions are used in Magix to traverse the Node structure and they can be thought of as XSLT or SQL queries, though much simpler in nature. Every expression starts with a '{' and ends with a '}'. Between these is the actual expression. The expression might be relative, meaning it starts with an '[' immediately after the '{'. Or the expression can be absolute, by using 'root' or 'DataSource' as its first keyword. Sometimes other types of logic, such as 'foreach' in the executor will add other values which can be de-referenced. For 'foreach' you can also use 'idx' in addition to 'root' and relative expressions. Here is an expression for you as an example; '{[Object][Name].Value}'. The previous expression will return the Value of the node which must be found relatively within [Object] node within its [Name] node. Expressions can end with .Value, .Name or ']'. The latter will return a list of nodes. Not all places that can handle expressions can handle all these three different results. For instance, 'foreach' needs the expression to end with ']' to make sure a Node List is returned. .Value returns the Value of the node from the expression, .Name its name and ']' the Node list of the given child. Integer numbers can also be used for cases where you do not know the name of the node you are traversing. For instance '{root[Objects][0].Value}' will return the root parent node's [Object] node's first child's Value. All counting with computers starts with 0, so 0 equals 1st item. In addition to Named and Numbered nodes, you can also go one step up by using e.g. '{[../][Name].Value}'. The previous expression will go one up, from current context, find the [Name] node and return its Value.

```
[ActiveEvent(Name = "Magix.Common.ReplaceStringLiteral")]
protected void Magix_Common_ReplaceStringLiteral(object sender, EventArgs e)
```

Level2: Will replace all occurrences of 'Replace' Value, which must be an Expression with the expression in 'Replacement' Value node, within the 'Source' expression. For instance, if you have a node in your DataSource at DataSource[Object][Properties][Email].Value containing; "sdfouh sdfouh sdfu sdfiqqweeqw !!Name!! mumbo-jumbo" and you want to replace all occurrences of the string literal '!!Name!!' with whatever happens to be in your current DataSource[Name][0].Value, then the Node structure you'd have to create to pass into this Event is; [Replace].Value equals '!!Name!!' [Replacement].Value equals '{[Name][0].Value}' [Source].Value = '[Object][Properties][Email].Value' Both 'Replace' and 'Replacement' might either be static text or Expressions. All three parameters must somehow point to single instances, and not Node lists or anything, but they can all end with either 'Value' or 'Name'

```
[ActiveEvent(Name = "Magix.Common.SendEmailFromForm")]
protected void Magix_Common_SendEmailFromForm(object sender, EventArgs e)
```

Level1: Will serialize the Active SingleView Form and send an email from 'Email' and 'From' to the 'Email' field on the given form. Before the email is sent both the 'Header' and the 'Body' will substitute every single occurrence of [x] with the MetaView's property with the same name [x]

```
[ActiveEvent(Name = "Magix.Common.ReplaceStringValue")]
protected void Magix_Common_ReplaceStringValue(object sender, EventArgs e)
```

Level2: Will do a String.Replace on the given 'Source' or 'SourceNode'. Will replace 'OldString' or 'OldStringNode' with 'NewString' or 'NewStringNode' and return the value either in 'Result' or 'ResultNode', direct value [no 'Node' part] always have preference

```
[ActiveEvent(Name = "Magix.Common.MultiAction")]
protected void Magix_Common_MultiAction(object sender, EventArgs e)
```

Level2: Will call 'Magix.MetaAction.RaiseAction' for every single 'ActionName' in the Actions [list] Parameter. Useful for creating complex abstractions, doing multiple tasks at once or 'encapsulating' your entire logic inside one Action

```
[ActiveEvent(Name = "Magix.Common.RenameNode")]
protected void Magix_Common_RenameNode(object sender, EventArgs e)
```

Level2: Will rename the given 'FromName' to 'ToName'. Will throw exception if no 'FromName' exists, or parameters are missing

```
[ActiveEvent(Name = "Magix.Common.StripAllParametersExcept")]
protected void Magix_Common_StripAllParametersExcept(object sender, EventArgs e)
```

Level2: Will strip every single Parameter OUT of the Node structure except the given 'But'. But can be either one single name of an object or a list of nodes containing several names. Useful for shrinking nodes as they grow due to being passed around or being parts of MultiActions or something similar

```
[ActiveEvent(Name = "Magix.Common.GetSingleMetaObject")]
protected void Magix_Common_GetSingleMetaObject(object sender, EventArgs e)
```

Level2: Will return the given MetaObject [ID] as a Key/Value pair. Will not traverse Child Objects though. Useful for fetching objects for any one reason you might have, as long as you know their ID

```
[ActiveEvent(Name = "Magix.Common.ReloadOriginalWebPart")]
protected void Magix_Common_ReloadOriginalWebPart(object sender, EventArgs e)
```

Level2: Will reload the Original WebPart, intended to be, within the 'current WebPart container' on the page. Meaning, if you've allowed the user to 'fuzz around all over the place' till he no longer can remember what originally was within a specific WebPart Container, he can raise this event [somehow], which will 'reload the original content' into the 'current container' [container raising the event]

```
[ActiveEvent(Name = "Magix.Common.LoadSignatureForCurrentMetaObject")]
protected void Magix_Common_LoadSignatureForCurrentMetaObject(object sender,
    EventArgs e)
```

Level2: If raised from within a MetaView on a specific MetaObject ['ID'], somehow, will show the SignatureModule for that particular MetaObject for its 'ActionSenderName' property. When Signature is done [signing complete] the original content of the Container will be reloaded

```
[ActiveEvent(Name = "Magix.MetaView.UnLoadSignature")]
protected void Magix_Signature_UnLoadSignature(object sender, EventArgs e)
```

Level2: Helper for SignatureModule, to store it correctly upon finishing and saving a new Signature. Will extract the 'Signature' content and store into the 'Name' property of the given 'ID' MetaObject and save the MetaObject

```
[ActiveEvent(Name = "Magix.Common.SetSessionVariable")]
protected void Magix_Common_SetSessionVariable(object sender, EventArgs e)
```

Level2: Will set the given Session Variable ['Name'] to the 'Value'. Useful for creating caches of huge things, you need to occur really fast [or something]. Session Variables like these can later be retrieved by its sibling method 'Magix.Common.GetSessionVariable'. Things stored into the Session will be on a per user level [meaning, it'll take a LOT of memory on your server], but it will be very fast to retrieve later. Be Cautious here!

```
[ActiveEvent(Name = "Magix.Common.GetSessionVariable")]
protected void Magix_Common_GetSessionVariable(object sender, EventArgs e)
```

Level2: Will return the given Session Variable ['Name'] to the 'Value' output node. Useful for retrieving caches of huge things, you need to occur really fast [or something]. Session Variables like these can be set by its sibling method 'Magix.Common.SetSessionVariable'. Things stored into the Session will be on a per user level [meaning, it'll take a LOT of memory on your server], but it will be very fast to retrieve later. Be Cautious here!

```
[ActiveEvent(Name = "Magix.Common.ExportMetaView2CSV")]
protected void Magix_Common_ExportMetaView2CSV(object sender, EventArgs e)
```

Level2: Expects to be given a 'MetaViewName' which it will turn into a MetaView object, which it will use as its foundation for exporting all MetaObjects of the MetaView's TypeName into a CSV file, which it will redirect the client's web browser towards. You can also override how the type is being rendered by adding up 'WhiteListColumns' and 'Type' parameters, which will override the default behavior for the MetaView. Set 'Redirect' to false if you wish to stop redirecting to the newly created file to occur. Regardless the relative path to the file created will be returned as 'FileName' and the number of records as 'NoRecords'

```
[ActiveEvent(Name = "Magix.Common.ExportMetaViewObjectList2CSV")]
protected void Magix_Common_ExportMetaViewObjectList2CSV(object sender, EventArgs e)
```

Level2: Will export a node list in 'Objects' List form to a CSV file, and redirect the client to that newly created CSV file. Set 'Redirect' to false to stop redirecting to the newly created CSV file to occur

```
[ActiveEvent(Name = "Magix.Common.RedirectClient")]
protected void Magix_Common_RedirectClient(object sender, EventArgs e)
```

Level2: Redirect clients to the given 'URL' parameter

```
[ActiveEvent(Name = "Magix.Common.ImportCSVFile")]  
protected void Magix_Common_ImportCSVFile(object sender, EventArgs e)
```

Level2: Given a 'FileName' a 'Folder' and a 'MetaViewName' this method will try to import the given CSV file according to the given MetaView ['MetaViewName'] into your MetaObject data storage

```
public object GetExpressionValue(string expression, Node source)
```

Level2: Documented, although not directly accessible for anyone outside of code, though used heavily in virtually every single module and component which uses Expressions, which are statements of text starting with '{', without the quotes. This method will return one out of four different possible values, a null value, meaning 'empty value', as in for instance; 'not found' or 'didn't have a value'. It can also return itself, if you try to check a text literal for its expression value, and it contains no expression, it will return 'itself' such as if for instance your expression parameter was for instance; 'Hello World...', it would return; 'Hello World...'. If your expression starts with a '{', it will be treated as a 'lookup' into your given source Node structure, for instance the expression; {[Objects][0][Email].Value} will return whatever happens to be at the Value property of the 'Email' node, which is in the 'first child' of the 'Objects' node within your DataSource, if you're in a Form of some sort for instance. While; {[0].Name} will return the name of the first node, and {[Objects]} will return all nodes in the 'Objects' node. If the expression starts with '{DataSource[', it will traverse and starts its traversal process at the outer most parent Node, the first Node within its Hierarchy

30. CreateDefaultInitialActions_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaTypes.CreateDefaultInitialActions_Controller

Description:

Level2: Contains Application Startup code to create the default Actions unless they're already there

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Will create some default installation Actions for the End User to consume in his own Meta Applications. These are all prefixed with 'Magix.DynamicEvent'

31. MetaView_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.MetaViews.MetaView_Controller

Description:

Level2: Contains helper logic for viewing and maintaing MetaViews, and related subjects. MetaViews are the foundation for the whole viewing parts of the Meta Application system. A MetaView is imperative for both being able to collect new data and also for viewing existing data. The MetaView defines which parts of the object you can see at any time too, which means you can use it to filter access according to which Grid the user is having access to, for instance. This controller contains logic for editing and maintaining MetaViews, plus also direct usage of MetaViews

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Handled to override the DeleteObject method from DBAdmin ...

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]
protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, EventArgs e)

Level 2: Returns the Desktop Icon for launching Actions back to caller

[ActiveEvent(Name = "Magix.Publishing.GetPluginMenuItems")]
protected void Magix_Publishing_GetPluginMenuItems(object sender, EventArgs e)

Level2: Will return one item back to caller which hopefully will function as the basis of loading the ViewMetaViews logic

```
[ActiveEvent(Name = "Magix.MetaView.ViewMetaViewMultiMode")]
protected void Magix_MetaType_ViewMetaViewMultiMode(object sender, EventArgs e)
```

Level2: Will show the given MetaView ['MetaViewName'] in MultiView mode. As in, the end user will see a Grid of all MetaObjects of the TypeName of the MetaView

```
[ActiveEvent(Name = "Magix.MetaView.ViewMetaViews")]
protected void Magix_MetaView_ViewMetaViews(object sender, EventArgs e)
```

Level2: Will show a Grid with all the MetaViews to the end user

```
[ActiveEvent(Name = "Magix.MetaView.MetaView_Single.GetTemplateColumnSelectView")]
protected void Magix_MetaView_MetaView_Single_GetTemplateColumnSelectView(object sender, EventArgs e)
```

Level3: Will return a SelectList with all the MetaViews back to caller

```
[ActiveEvent(Name = "Magix.MetaView.MetaView_Multiple.GetTemplateColumnSelectView")]
protected void Magix_MetaView_MetaView_Multiple_GetTemplateColumnSelectView(object sender, EventArgs e)
```

Level3: Will return a SelectList with all the MetaViews back to caller

```
[ActiveEvent(Name = "Magix.Meta.GetCopyMetaViewTemplateColumn")]
protected void Magix_Meta_GetCopyMetaViewTemplateColumn(object sender, EventArgs e)
```

Level3: Creates a Copy MetaView LinkButton

```
[ActiveEvent(Name = "Magix.Meta.CopyMetaView")]
protected void Magix_Meta_CopyMetaView(object sender, EventArgs e)
```

Level2: Will copy [deep clone] the incoming 'ID' MetaView and return the new copy's ID as 'NewID'

```
[ActiveEvent(Name = "Magix.Meta.CopyMetaViewAndEditCopy")]
protected void Magix_Meta_CopyMetaViewAndEditCopy(object sender, EventArgs e)
```

Level2: Will copy [deep clone] the incoming 'ID' MetaView and edit it immediately


```
[ActiveEvent(Name = "Magix.MetaView.DeleteMetaView")]
protected void Magix_MetaView_DeleteMetaView(object sender, EventArgs e)
```

Level2: Will ask the user if he really wish to delete the MetaView given through 'ID', and if the user says yes, delete it

```
[ActiveEvent(Name = "Magix.MetaView.DeleteMetaView- Confirmed")]
protected void Magix_MetaView_DeleteMetaView_Confirmed(object sender, EventArgs e)
```

Level2: End user confirmed he wishes to delete the MetaView

```
[ActiveEvent(Name = "Magix.MetaView.CreateMetaView")]
protected void Magix_MetaView_CreateMetaView(object sender, EventArgs e)
```

Level2: Will create a new MetaView with some default settings and return ID of new view as 'NewID'

```
[ActiveEvent(Name = "Magix.MetaView.CreateMetaViewAndEdit")]
protected void Magix_MetaView_CreateMetaViewAndEdit(object sender, EventArgs e)
```

Level2: Will create and Edit immediately a new MetaView

```
[ActiveEvent(Name = "Magix.MetaView.EditMetaView")]
protected void Magix_MetaView_EditMetaView(object sender, EventArgs e)
```

Level2: Will edit the given ['ID'] MetaView with its properties and some other controls. Such as for instance a 'View WYSIWYG' button

```
[ActiveEvent(Name = "Magix.MetaView.GetMetaViewActionTemplateColumn")]
protected void Magix_MetaView_GetMetaViewActionTemplateColumn(object sender,
    EventArgs e)
```

Level3: Will show all Actions for specific Meta View Property

```
[ActiveEvent(Name = "Magix.MetaView.ShowAllActionsForMetaViewProperty")]
protected void Magix_MetaView_ShowAllActionsForMetaViewProperty(object sender,
    EventArgs e)
```

Level2: Will show a Window with all the events for the specific Meta View Property, such that the end user can add and/or remove actions from the list

```
[ActiveEvent(Name = "Magix.MetaView.OpenAppendNewActionDialogue")]
protected void Magix_MetaView_OpenAppendNewActionDialogue(object sender,
ActiveEventArgs e)
```

Level2: Will show the 'List of Actions form' for appending and editing and deleting actions associated with the specific Action on the specific Meta View's Property

```
[ActiveEvent(Name = "Magix.MetaView.ActionWasSelected")]
protected void Magix_MetaView_ActionWasSelected(object sender, ActiveEventArgs e)
```

Level2: Will append the given 'ID' Action into the given 'ParentPropertyName' Event name and save the MetaForm.Node

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode_2(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the list data for editing Actions for form element on grid system

```
[ActiveEvent(Name = "Magix.MetaView.RemoveActionFromActionList")]
protected void Magix_MetaView_RemoveActionFromActionList(object sender, ActiveEventArgs e)
```

Level2: Sink for deleting Meta View Action from Event

```
[ActiveEvent(Name = "Magix.MetaView.AppendAction")]
protected void Magix_MetaView_AppendAction(object sender, ActiveEventArgs e)
```

Level2: Will show a Search box, from which the end user can search for a specific action to append to the list of actions already associated with the MetaView Property

```
[ActiveEvent(Name = "Magix.MetaView.ActionWasChosenForAppending")]
protected void Magix_MetaView_ActionWasChosenForAppending(object sender, ActiveEventArgs e)
```

Level2: Will append the specific 'ActionName' into the 'ParentID' MetaView Property and call for an update of the grids

```
[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]
protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender,
ActiveEventArgs e)
```

Level2: Handled to make sure we clear content4 and out if a MetaView s deleted.

```
[ActiveEvent(Name = "Magix.MetaView.ShowMetaViewMultipleInCurrentContainer")]
protected void Magix_MetaView_ShowMetaViewMultipleInCurrentContainer(object sender,
ActiveEventArgs e)
```

Level2: Will show the 'MetaViewName' MetaView within the 'current container'

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode(object sender, ActiveEventArgs e)
```

Level2: Sink for getting the type information for editing Actions for Meta Views

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectTypeNode")]
protected void DBAdmin_DynamicType_GetObjectTypeNode_2(object sender, ActiveEventArgs
e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode', front-web style

```
[ActiveEvent(Name = "DBAdmin.DynamicType.CreateObject")]
protected void DBAdmin_DynamicType_CreateObject(object sender, ActiveEventArgs e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode'

```
[ActiveEvent(Name = "DBAdmin.DynamicType.GetObjectsNode")]
protected void DBAdmin_DynamicType_GetObjectsNode(object sender, ActiveEventArgs e)
```

Level2: Overridden to handle MetaView 'dynamically displayed'. Meaning in "-META" 'mode'

```
[ActiveEvent(Name = "Magix.MetaView.MultiViewActionItemTemplateColumn")]
protected void Magix_MetaView_MultiViewActionItemTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will return a Button control to caller, upon which clicked, will raise the named actions in its settings according to which MetaViewProperty it belongs to. Front-web stuff, basically a MetaView form field with 'Actions' associated with it

```
[ActiveEvent(Name = "Magix.MetaView.RunActionsForMetaViewProperty")]  
protected void Magix_MetaView_RunActionsForMetaViewProperty(object sender,  
ActiveEventArgs e)
```

Level2: Will run the Actions associated with the MetaViewProperty given through 'MetaViewName' [MetaView - Name], 'Name' [of property within MetaObject] and expects to be raised from within a WebPart, since it will pass along the 'current container' onwards

```
[ActiveEvent(Name = "Magix.MetaView.MetaView_Multiple_GetColonTemplateColumn")]  
protected void Magix_MetaView_MetaView_Multiple_GetColonTemplateColumn(object sender,  
ActiveEventArgs e)
```

Level3: Will create a control type depending upon the colon-prefix of the column. For instance, if given date:When it will create a Calendar, putting the Value selected into the 'When' property. If given select:xx.yy:zz it will create a select list, enumerating into the ObjectTypes given with the Property de-referenced. E.g. select:Gender.Sex:Male-Female will enumerate every 'Sex' value of every object of TypeName 'Gender' and put it into the property with the name of 'Male-Female'. choice:Gender.Sex,Css:Male-Female would function identically, except it also expects to find a Css property, whos value will be used as the CSS class for a small 'clickable enumerating' type of control which would allow for 'single-choice' selection of status for instance. If you want to create plugin types for the MultiView Grid system, this event is what you'd have to handle to inject your own Column types

```
[ActiveEvent(Name = "Magix.MetaView.MetaView_Single_GetColonTemplateColumn")]
protected void Magix_MetaView_MetaView_Single_GetColonTemplateColumn(object sender,
ActiveEventArgs e)
```

Level3: Will create a; 'linkedE2M' control type depending upon the colon-prefix of the column. For instance, if given linkedE2M:Email:Name it will create a linked textbox, which is linked in its 'Name' field, which is linked towards another field on the same form, called 'Email', such that when the Email is typed into the Email field, Magix will automatically try to parse the name of the person from his email address. Or b; 'init-actions' which will basically just call all Actions in Property during the initial load of the form. This is where you'd like to put stuff such as initialization, maybe inclusion of custom CSS files, etc. Or c; autocompleter, which will need you to point to a table and a property name within that table, e.g. autocompleter:Customer.Name:Name will do a lookup into the Customer Meta Objects, find all Customers that have a 'Name' property which contains the search query in the textbox, and return as a list for the end-user to choose from, think Facebook search. If you type e.g. 'hans' into the textbox, it'll return all hansen, johansen and so on as items for the user to select. It will never show more than 10 items at the time. PS! Although tempting, do NOT USE the autocompleter for huge tables, meaning Meta Objects which you've got more than 500 items of. Due to some restrictions in the current internal algorithms, such a thing would make your application monstrously slow. If there are Actions associated with an autocomplater, they will be raised with the user selects an item from the drop down list, and only then. If you want to add an Image to the AutoCompleter items, and there's a Property pointing to an Image URL within the objects you're querying for within your AutoCompleter, then you can add a comma [,] and then the name of the property that contains the URL to the image which you want to use as the 'Avatar' for the AutoCompleter item

```
[ActiveEvent(Name = "Magix.MetaView.LoadWysiwyg")]
protected void Magix_MetaView_LoadWysiwyg(object sender, EventArgs e)
```

Level2: Loads up WYSIWYG editor for MetaView in 'SingleView Mode'

```
[ActiveEvent(Name = "Magix.MetaView.GetViewData")]
protected void Magix_MetaView_GetViewData(object sender, EventArgs e)
```

Level2: Returns the properties for the MetaView back to caller

```
[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,
ActiveEventArgs e)
```

Level2: Returns the number of MetaViews and an event for viewing all MetaViews back to caller

```
[ActiveEvent(Name = "Magix.MetaView.CreateSingleViewMetaObject")]  
protected void Magix_MetaView_CreateSingleViewMetaObject(object sender, EventArgs e)
```

Level2: Will create a new MetaObject according to the values given from the MetaView_SingleView form. 'PropertyValues' is expected to contain a Name/Value list-pair, and the ViewName is supposed to be the unique name to the specific MetaView being used.

```
[ActiveEvent(Name = "DBAdmin.Data.DeleteObject- Override")]  
protected void DBAdmin_Data_DeleteObject_Override(object sender, EventArgs e)
```

Level2: Overridden to get positioning correct ...

32. PDF_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.PDF.PDF_Controller

Description:

Level2: Contains logic for creating PDF files and downloading to Client or saving locally

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.PDF.CreatePDF- Book")]

protected void Magix_PDF_CreatePDF_Book(object sender, ActiveEventArgs e)

Level2: Will create a PDF document and redirect the user to that document as named in the 'File' parameter. Takes many different parameters; 'FrontPage' [optional] being an image [600x846 px big] to a cover page you'd like to use. 'Index' [optional] containing all components needed to build the books 'index'. 'Chapters' contains a list of all the pages, with HTML parsing capabilities [one level only]

33. Logging_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Logging_Controller

Description:

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.ViewLog")]

private void Magix_Publishing_ViewLog(object sender, EventArgs e)

Level2: Will display the Grid of the latest Log Items for the user to be able to drill down, and have a more thorough look

[ActiveEvent(Name = "Magix.Publishing.GetLogItems")]

private void Magix_Publishing_GetLogItems(object sender, EventArgs e)

Level2: Will return the last log items according to newest first

[ActiveEvent(Name = "Magix.Publishing.ViewLogItem")]

protected void Magix_Publishing_ViewLogItem(object sender, EventArgs e)

Level2: Will show one LogItem in a grid for the end-user to scrutinize. Will allow the user to always have to LogItems up at the same time

34. LoginOut_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.LoginOut_Controller

Description:

Level2: Login and Logout Controller

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.LogInUser")]
protected void Magix_Core_LogInUser(object sender, EventArgs e)

Level2: Basically just checks to see if the given 'Username'/'Password' combination exists, and if so sets the User.Current object, which is static per request, returning the logged in user. Then if successful, raises the 'Magix.Core.UserLoggedIn' event. Normally this implies a re-direct of the entire page, going to the 'root page level' automatically if the login attempt was successful

[ActiveEvent(Name = "Magix.Core.UserLoggedIn")]
protected void Magix_Core_UserLoggedIn(object sender, EventArgs e)

Level2: Redirects the user to Root, unless another Redirect path is somehow given

[ActiveEvent(Name = "Magix.Publishing.GetStateForLoginControl")]
protected void Magix_Publishing_GetStateForLoginControl(object sender, EventArgs e)

Level2: Returns whether or not the User is logged in or not for the LoginInOut Module to know what types of controls to load

```
[ActiveEvent(Name = "Magix.Core.UserLoggedOut")]  
private void Magix_Core_UserLoggedOut(object sender, EventArgs e)
```

Level2: Resets the User.Current object, and redirects user back to root

35. FileSystem_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.FileSystem_Controller

Description:

Level2: Tiny helper for menu item in Administrator Dashboard to view the File System Browser

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.ViewFileSystem")]

protected void Magix_Publishing_ViewFileSystem(object sender, EventArgs e)

Level2: Will call 'Magix.FileExplorer.LaunchExplorer' with a couple of predefined values for positioning and such. Clears all containers from 4 and out

36. InitialLoading_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.InitialLoading_Controller

Description:

Level2: Class for taking care of the Magix.Core.InitialLoading message, meaning the initialization of the page on a per User level initially as they're coming to a new 'page' or URL ... Basically just handles Magix.Core.InitialLoading and mostly delegates from that point ...

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.InitialLoading")]

protected void Magix_Core_InitialLoading(object sender, ActiveEventArgs e)

Level2: This method will handle the 'initial loading', meaning basically that the page was loaded initially by either changing the URL of the browser to the app or doing a Refresh or something. For coders, meaning basically not IsPostBack on the page object ... Throws a whole range of different events based upon whether or not the User is logged in or not, and which URL is being specifically requested. Its most noticable outgoing event would though be 'Magix.Publishing.LoadDashboard' and 'Magix.Publishing.UrlRequested'. PS! If you override this one, you've effectively completely bypassed every single existing logic in Magix, and everything are 'dead event handlers' in 'limbo' not tied together at all. Which might be cool, or might be a nightmare, depending upon how you use it, if you use it, use it with CARE if you do though !!

37. SliderMenu_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.SliderMenu_Controller

Description:

Level2: Helps feed the SliderMenu in Front-Web to get its Items to build its structure upon

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetSliderMenuItems")]

protected void Magix_Publishing_GetSliderMenuItems(object sender, EventArgs e)

Level2: Will return a node containing all the menu items in your Pages hierarchy, possibly according to how you've got access to them, as long as the Access Controller is in no ways jeopardized

[ActiveEvent(Name = "Magix.Publishing.SliderMenuItemClicked")]

protected void Magix_Publishing_SliderMenuItemClicked(object sender, EventArgs e)

Level2: Will Open the specific WebPage accordingly to which SlidingMenuItem was clicked

private void GetOneMenuItem(Node node, WebPage po, bool isRoot)

Level2: Will build up the Node structure for one Menu item, first verifying the User has access to that particular page by raising the 'Magix.Publishing.GetRolesListForPage' event

38. TipOfToday_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.TipOfToday_Controller

Description:

Level2: Creates our default Tooltips or Tutorial rather

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, ActiveEventArgs e)

Level2: Creates our default tooltips if there exists none

39. OpenID_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.OpenID_Controller

Description:

Level2: Controller implementing the OpenID logic for both Relying Party and OpenID Provider logic. Every User in Magix can use the website where Magix is installed as an OpenID provider by typing in the root URL of the web application and append ?openID=username In addition you can associate as many OpenID tokens as you wish to your User and then use these other OpenID providers to log into Magix.

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Overriding our default Magix Login logic here to inject our own OpenID stuff ... Overrides; 'Magix.Core.LogInUser' and 'Magix.Core.UserLoggedIn' which again are being used by the Magix 'core' login logic, meaning we're basically doing effectively 'AOP' here by overriding existing events like these ... Where the 'Aspect' here would be the OpenID 'logic'

[ActiveEvent(Name = "Magix.Core.InitialLoading")]
protected void Magix_Core_InitialLoading(object sender, EventArgs e)

Level2: Basically just checking if this is an OpenID Request, and doing either the Provider or the Relying Party thing needed ...

```
[ActiveEvent(Name = "Magix.Publishing.NewUserRegisteredThroughOpenID")]
protected void Magix_Publishing_NewUserRegisteredThroughOpenID(object sender,
ActiveEventArgs e)
```

Level2: Creates a default User object, and adds him into the default role as set through the 'Magix.Publishing.OpenID.DefaultRoleName' setting. Associates the given OpenID Token with the User [obviously] Will also try to extract additional information, such as Address etc from the OpenID Provider ...

```
[ActiveEvent(Name = "Magix.Core.LogInUser- Override")]
protected void Magix_Core_LogInUser_Override(object sender, EventArgs e)
```

Level2: Overridden to check if we're in OpenID Mode or in Username/Password mode Will forward to overridden method if no OpenID value is given ...

```
[ActiveEvent(Name = "Magix.Core.UserLoggedIn- Override")]
protected void Magix_Core_UserLoggedIn_Override(object sender, EventArgs e)
```

Level2: Overridden to check to see if we're an OpenID Provider, and if so, redirect back to Relying Party accordingly. If we're not a Provider, it will forward the event to the overridden logic

```
private void InjectOpenIDHeaderHTMLMetaElements()
```

Helper method used to inject OpenID Meta information into the header element of the Page.

```
private void ProcessAnyOpenIDProviderRequests()
```

Will try to extract DotNetOpenAuth OpenID requests being passed around, and process them, if any

```
private void ProcessAnyOpenIDRelyingPartyRequests()
```

Will check to see if there's any OpenID Relying Party requests currently coming in, or in the pipeline, and process them if necessary.


```
private void OpenIDProviderVerifiedToken(IAuthenticationResponse r)
```

OpenID Token was verified, can we log in directly, or are we supposed to create a new User object. As in is this OpenID Token associated with an existing User here ...?

```
private bool AuthenticateUserFromIncomingOpenID(IAuthenticationResponse r)
```

Will either log in existing user from an OpenID Token and return true, or not be able to match him with an existing user and hence return false

```
private void LogInExistingUserFromOpenIDToken(IAuthenticationResponse r, OpenIDToken token)
```

Will log in User from an existing OpenID Token, and show a message box back to the end user to show him we had success

```
private void CreateNewUserFromOpenIDToken(IAuthenticationResponse r)
```

Will raise the 'Magix.Publishing.NewUserRegisteredThroughOpenID' with all the data given from the OpenID Provider. And show the end user a dialog confirming he's successfully authenticated

40. PageLoader_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.PageLoader_Controller

Description:

Level2: Controller responsible for loading up 'Page Objects', and doing initialization around pages and such as they're being loaded

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.UrlRequested")]

protected void Magix_Publishing_UrlRequested(object sender, EventArgs e)

Level2: Basically just 're-maps' the event to Magix.Publishing.OpenPage and changes the incoming URL parameter to an outgoing ID parameter of the specific page being requested ...

[ActiveEvent(Name = "Magix.Publishing.OpenPage")]

protected void Magix_Publishing_OpenPage(object sender, EventArgs e)

Level2: Expects an ID parameter which should be the ID of a Page. Will loop through every WebPart of the page, and raise 'Magix.Publishing.InjectPlugin' for every WebPart within the WebPage object. Might throw exceptions if the page is not found, or the user has no access rights to that specific page or something ... Will verify the user has access to the page, by raising 'Magix.Publishing.VerifyUserHasAccessToPage' and if no access is granted, try to find the first child object the user has access to through raising the 'Magix.Publishing.FindFirstChildPageUserCanAccess' event. Will throw SecurityException or ArgumentException if page not found or granted access to [nor any of its children]

```
[ActiveEvent(Name = "Magix.Core.GetContainerForControl")]  
protected void Magix_Core_GetContainerForControl(object sender, EventArgs e)
```

Level2: Will return the Container ID of the 'Current Container', meaning whomever raised whatever event we're currently within. Meaning if you've got a Button in a Grid which raises some event which is dependent upon knowing which Container its being raised from within, to load some other control [e.g. Signature Column in Grid or Edit Object] Is dependent upon that the PageObjectTemplateID is being passed around some way or another into this bugger

```
private void IncludeStaticFrontEndCSS()
```

Includes static CSS for our front-end

```
private void OpenPage(WebPage page)
```

Helper method used in 'Magix.Publishing.OpenPage' for opening the page once it has been granted access to either directly, or one of its child pages. Will basically just loop through every WebPart in the Page and raise 'Magix.Publishing.InjectPlugin' for each of them, after doing some manipulation of the Container the WebPart is supposed to become injected into and such. Will also set the title of the page according to the name of the WebPage

```
private void SetCaptionOfPage(WebPage page)
```

Sets the title of the page by raising 'Magix.Core.SetTitleOfPage' event according to the Name of the Page

41. WebPart_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.WebPart_Controller

Description:

Level2: Initializes WebParts and such while being injected onto WebPage

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.InitializePublishingPlugin")]

protected void Magix_Publishing_InitializePublishingPlugin(object sender, ActiveEventArgs e)

Level2: Initializes the Plugin with its values according to what's in its settings

[ActiveEvent(Name = "Magix.Publishing.ReloadWebPart")]

protected void Magix_Publishing_ReloadWebPart(object sender, ActiveEventArgs e)

Level2: Reloads the Original WebPart content

[ActiveEvent(Name = "Magix.Publishing.GetWebPartSettingValue")]

private void Magix_Publishing_GetWebPartSettingValue(object sender, ActiveEventArgs e)

Level2: Returns the Value of a specific setting of a specific WebPart

[ActiveEvent(Name = "Magix.Publishing.InjectPlugin")]

private void Magix_Publishing_InjectPlugin(object sender, ActiveEventArgs e)

Level2: Injects one plugin into the given container, unless anything says the Plugin doesn't need to be reloaded for some reasons [sliding menu for instance]

42. AdministratorDashboard_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.AdministratorDashboard_Controller

Description:

Level2: Main 'router' in dispatching important [ADMIN] Dashboard functionality [As in; Administrator user logged in]

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.LoadAdministratorDashboard")]

protected void Magix_Publishing_LoadAdministratorDashboard(object sender, EventArgs e)

Level2: Loads Administrator Dashboard [back-web]

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]

protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, EventArgs e)

Level2: Will return the icons needed to populate the 'desktop' with your most important 'shortcut icons'

[ActiveEvent(Name = "Magix.Publishing.DownloadHeavyDocumentation")]

protected void Magix_Publishing_DownloadHeavyDocumentation(object sender, EventArgs e)

Level2: Will redirect the browser towards the complete PDF documentation for Magix', including all the C# and coding parts

```
[ActiveEvent(Name = "Magix.Publishing.LoadAdministratorMenu")]  
protected void Magix_Publishing_LoadAdministratorMenu(object sender, EventArgs e)
```

Level2: Loads the Administrator SlidingMenu into the 1st content, but everything here is basically overridable. Will also raise 'Magix.Publishing.GetPluginMenuItems' to allow for plugins to connect up their own Administrator Dashboard menu items

```
[ActiveEvent(Name = "Magix.Publishing.GetDataForAdministratorDashboard")]  
protected void Magix_Publishing_GetDataForAdministratorDashboard(object sender,  
ActiveEventArgs e)
```

Level2: Ads up the 'common data' for the Admin Dashboard such as Users, Roles etc

```
[ActiveEvent(Name = "Magix.Publishing.ViewClasses")]  
private void Magix_Publishing_ViewClasses(object sender, EventArgs e)
```

Level2: Will fire up our Database Manager

43. ChildExcerpt_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.ChildExcerpt_Controller

Description:

Level2: Helps out with some of the editing and using of the ChildExcerpt Module Type

Basetypes

Magix.Brix.Loader.ActiveController

Methods

```
[ActiveEvent(Name = "Magix.Publishing.GetEditChildExcerptNoArticlesDropDown")]
private void Magix_Publishing_GetEditChildExcerptNoArticlesDropDown(object sender,
ActiveEventArgs e)
```

Level3: Creates a SelectList from which the no of articles in the ChildExcerpt Object can be chosen

```
[ActiveEvent(Name = "Magix.Publishing.BuildOneChildExcerptControl")]
private void Magix_Publishing_BuildOneChildExcerptControl(object sender, EventArgs e)
```

Level3: The actual construction of our Child Excerpt panels are being done here

```
[ActiveEvent(Name = "Magix.Publishing.GetLastChildrenPages")]
private void Magix_Publishing_GetLastChildrenPages(object sender, EventArgs e)
```

Level2: Will return the last 'Count' number of pages, sorted according to newest first

```
[ActiveEvent(Name = "Magix.Publishing.GetTemplateColumnSelectChildExcerptNo")]
protected void Magix_Publishing_GetTemplateColumnSelectChildExcerptNo(object sender,
ActiveEventArgs e)
```

Level3: Will create a SelectList for editing of the number of ChildExcerpts in the WebPart

44. EditUser_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditUser_Controller

Description:

Level2: Here mostly for Editing User, and also to some extent creating default users and roles upon startup if none exists

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Makes sure we have Administrator and User Roles, in addition creates a default user [admin/admin] if no user exists

[ActiveEvent(Name = "Magix.Core.SaveSettingsSection- Override")]
protected void Magix_Core_SaveSettingsSection_Override(object sender, EventArgs e)

Level2: Overridden to make sure setting sections are saved and loaded per user

[ActiveEvent(Name = "Magix.Core.LoadSettingsSection- Override")]
protected void Magix_Core_LoadSettingsSection_Override(object sender, EventArgs e)

Level2: Overridden to make sure setting sections are saved and loaded per user

[ActiveEvent(Name = "Magix.Publishing.EditUsers")]
protected void Magix_Publishing_EditUsers(object sender, EventArgs e)

Level2: Allows editing of all users, and searching and changing properties and such. Shows all Users in a Grid

```
[ActiveEvent(Name = "Magix.Publishing.EditUser")]  
protected void Magix_Publishing_EditUser(object sender, EventArgs e)
```

Level2: Edit one specific User, and all of his properties

```
[ActiveEvent(Name = "Magix.Publishing.CreateUser")]  
protected void Magix_Publishing_CreateUser(object sender, EventArgs e)
```

Level2: Callback for creating a New User from Dashboard that pops up editing the specific user after his initial creation

```
[ActiveEvent(Name = "Magix.Publishing.ChangeAvatarForUser")]  
protected void Magix_Publishing_ChangeAvatarForUser(object sender, EventArgs e)
```

Level2: User has requested a change of Avatar for either himself, or another user. Loads up the FileExplorer Selector, and allows for selecting a new avatar for the admin user

```
[ActiveEvent(Name = "Magix.Publishing.SelectImageForUser")]  
protected void Magix_Publishing_SelectImageForUser(object sender, EventArgs e)
```

Level2: New image was selected in our FileExplorer as a new Avatar for our User. Will change the User's Avatar and save it and update

```
[ActiveEvent(Name = "Magix.Publishing.GetDefaultAvatarURL")]  
protected void Magix_Publishing_GetDefaultAvatarURL(object sender, EventArgs e)
```

Level2: Callback for supplying the Default Avatar for a User. Override this one if you wish to change the default image for a new user, which might be useful in e.g. OpenID scenarios and profiling efforts

```
[ActiveEvent(Name = "Magix.Publishing.GetRoleTemplateColumn")]  
private void Magix_Publishing_GetRoleTemplateColumn(object sender, EventArgs e)
```

Level3: Get 'select Role' template column for our Grid. Creates a SelectList and returns back to caller. PS! We only allow for editing the user such that he or she can belong to only one Role. If you'd like to extend this logic to allow for more than one Role to be added, this is where you'd want to do such, probably by overriding this event, and choose some sort of 'multiple choice' control to return instead

45. AdminDashboardHeader_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.AdminDashboardHeader_Controller

Description:

Level2: We'll 'lock' the Header control since it can be VERY annoying sometimes due to the DB Manager, which seriously needs to be refactored here BTW ...

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.LoadAdministratorHeader")]

protected void Magix_Publishing_LoadAdministratorHeader(object sender, ActiveEventArgs e)

Level2: Basically just ensures the Header Menu is loaded, and locked with 'Administrator Dashboard' as its value.

46. EditRoles_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditRoles_Controller

Description:

Level2: Here mostly to serve up grids and such for editing of Roles in the system

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.EditRoles")]

protected void Magix_Publishing_EditRoles(object sender, ActiveEventArgs e)

Level2: Will allow the end-user to edit [add, change, delete] roles within the system

47. EditTemplates_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditTemplates_Controller

Description:

Level2: Controller for editing Templates. Contains all the relevant event handlers and logic for editing your templates

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.EditTemplates")]

protected void Magix_Publishing_EditTemplates(object sender, ActiveEventArgs e)

Level2: Shows the grid containing all your Templates such that you can edit them or create new Templates and such

[ActiveEvent(Name = "DBAdmin.Common.ComplexInstanceDeletedConfirmed")]

protected void DBAdmin_Common_ComplexInstanceDeletedConfirmed(object sender, ActiveEventArgs e)

Level2: Only here to make sure that if a WebPageTemplate is being deleted from the Grid while editing it [for instance], we clear from 'content4' and out to make sure we're not allowig for editing of a deleted Template

[ActiveEvent(Name = "Magix.Publishing.GetCssTemplatesForWebPartTemplate")]

protected void Magix_Publishing_GetCssTemplatesForWebPartTemplate(object sender, ActiveEventArgs e)

Level2: Will return all possible 'CSS Template Values' from the 'web-part-templates.css' file within the media/module/ folders

```
[ActiveEvent(Name = "Magix.Publishing.ChangeCssForWebPartTemplateFromCssTemplate")]
protected void Magix_Publishing_ChangeCssForWebPartTemplateFromCssTemplate(object
sender, ActiveEventArgs e)
```

Level2: Changes the CSS class of the WebPartTemplate instance according to the Selected Css value chosen

```
[ActiveEvent(Name = "Magix.Publishing.CopyTemplate")]
protected void Magix_Publishing_CopyTemplate(object sender, ActiveEventArgs e)
```

Level2: Completes a 'deep copy' of the Template and saves it and loads up the Editing of it for the user to immediately start editing it

```
[ActiveEvent(Name = "Magix.Publishing.EditTemplate")]
protected void Magix_Publishing_EditTemplate(object sender, ActiveEventArgs e)
```

Level2: Edits one specific WebPageTemplate by loading up the 'Magix.Brix.Components.ActiveModules.Publishing.EditSpecificTemplate' module

```
[ActiveEvent(Name = "Magix.Publishing.DeleteWebPartTemplate")]
protected void Magix_Publishing_DeleteWebPartTemplate(object sender, ActiveEventArgs e)
```

Level2: Deletes a specific WebPartTemplate

```
[ActiveEvent(Name = "Magix.Publishing.GetWebPageTemplates")]
protected void Magix_Publishing_GetWebPageTemplates(object sender, ActiveEventArgs e)
```

Level2: Returns a node of all WebPageTemplates in the system

```
[ActiveEvent(Name = "Magix.Publishing.GetPublisherPlugins")]
protected void Magix_Publishing_GetPublisherPlugins(object sender, ActiveEventArgs e)
```

Level2: Returns a node of all PublisherPlugins you've got available in your installation

```
[ActiveEvent(Name = "Magix.Publishing.ChangeModuleTypeForWebPartTemplate")]
protected void Magix_Publishing_ChangeModuleTypeForWebPartTemplate(object sender,
ActiveEventArgs e)
```

Level2: Changes the ModuleName [plugin type] of the WebPartTemplate and saves it

```
[ActiveEvent(Name = "Magix.Publishing.ChangeTemplateProperty")]
protected void Magix_Publishing_ChangeTemplateProperty(object sender, EventArgs e)
```

Level2: Changes properties of the Template and saves it

```
[ActiveEvent(Name = "Magix.Publisher.GetNoContainersTemplateColumn")]
protected void Magix_Publisher_GetNoContainersTemplateColumn(object sender,
    EventArgs e)
```

Level3: Creates the Container column [select list] for selecting different number of WebParts for your template

```
[ActiveEvent(Name = "Magix.Publisher.ChangeNumberOfContainersOnTemplates-
Confirmed")]
protected void Magix_Publisher_ChangeNumberOfContainersOnTemplates_Confirmed(object
    sender, EventArgs e)
```

Level2: Will change the number of WebParts in your WebPartTemplate

```
[ActiveEvent(Name = "Magix.Publisher.GetCopyTemplateColumn")]
private void Magix_Publisher_GetCopyTemplateColumn(object sender, EventArgs e)
```

Level3: Creates the 'Copy Template' LinkButtons and returns back to the Grid system so that we can have an implementation of 'Copy Template' button. LinkButton raises 'Magix.Publishing.CopyTemplate' when clicked

```
private void RaiseTryToChangeNumberOfWebParts(WebPageTemplate t, int count)
```

Level2: Checks to see if number of webparts are decreasing, if they are, warning user about that he should be careful, blah, blah, blah ...

48. Dashboard_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Dashboard_Controller

Description:

Level2: Main 'router' in dispatching important Dashboard functionality

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.LoadDashboard")]

protected void Magix_Publishing_LoadDashboard(object sender, ActiveEventArgs e)

Level2: Will load the Dashboard if User is of type Administrator through raising 'Magix.Publishing.LoadDashboard', otherwise raise 'Magix.Publishing.LoadUserDashboard' which isn't currently being caught any place

49. EditPages_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.EditPages_Controller

Description:

Level2: Contains logic for Editing WebPage objects and such for administrator

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]

protected void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Overridden to make sure we've got some default pages during startup if none exists from before

[ActiveEvent(Name = "Magix.Publishing.EditPages")]

protected void Magix_Publishing_EditPages(object sender, EventArgs e)

Level2: Will load a Tree of the Pages from which you can browse around within to edit and view the relationships between all your WebPage objects

[ActiveEvent(Name = "Magix.Publishing.GetEditPagesDataSource")]

protected void Magix_Publishing_GetEditPagesDataSource(object sender, EventArgs e)

Level2: Will return the WebPages in your system back to caller in the return value as 'Items' collections

[ActiveEvent(Name = "Magix.Publishing.EditSpecificPage")]

protected void Magix_Publishing_EditSpecificPage(object sender, EventArgs e)

Level2: Will edit one Specific page according to either SelectedItemID [Tree] or ID parameter

```
[ActiveEvent(Name = "Magix.Publishing.PageWasUpdated")]  
protected void Magix_Publishing_PageWasUpdated(object sender, EventArgs e)
```

Level2: Will make sure we update our Tree control

```
[ActiveEvent(Name = "Magix.Publishing.ChangePageProperty")]  
protected void Magix_Publishing_ChangePageProperty(object sender, EventArgs e)
```

Level2: Changes the given properties of the WebPage object. Legal values are Text, URL, PageTemplateID

```
[ActiveEvent(Name = "Magix.Publishing.CreateChildWebPage")]  
protected void Magix_Publishing_CreateChildWebPage(object sender, EventArgs e)
```

Level2: Will create a new WebPage being the child of the given 'ID' WebPage

```
[ActiveEvent(Name = "Magix.Publishing.DeletePageObject")]  
protected void Magix_Publishing_DeletePageObject(object sender, EventArgs e)
```

Level2: Will ask the end user if he wish to delete specific Page object and all of its children

```
[ActiveEvent(Name = "Magix.Publishing.DeletePageObject- Confirmed")]  
protected void Magix_Publishing_DeletePageObject_Confirmed(object sender, EventArgs e)
```

Level2: Will delete specific Page object ['ID'] and all of its children

```
[ActiveEvent(Name = "Magix.Publishing.ChangeWebPartSetting")]  
protected void Magix_Publishing_ChangeWebPartSetting(object sender, EventArgs e)
```

Level2: Will update and save the incoming WebPartID [WebPartSetting] and set its Value property to 'Value'

```
[ActiveEvent(Name = "Magix.Publishing.WebPartTemplateWasModified")]
protected void Magix_Publishing_WebPartTemplateWasModified(object sender, EventArgs e)
```

Level3: Will run through all WebParts which are 'touched' by this WebPartTemplate and update their settings according to whatever new module was chosen. Warning; This will set any 'value properties' in these WebParts to their default value. It might also retrieve values it had before if this WebPart has earlier been this type of WebPartTemplate

```
[ActiveEvent(Name = "Magix.Publishing.WebPageTemplateWasModified")]
protected void Magix_Publishing_WebPageTemplateWasModified(object sender, EventArgs e)
```

Level2: Will update all WebPages that was modified by changing the WebPageTemplate. This might include creating new WebParts with default values, or removing existing WebParts on Pages

```
[ActiveEvent(Name = "Magix.Core.GetHyperLinkURL")]
protected void Magix_Core_GetHyperLinkURL(object sender, EventArgs e)
```

Level2: Will open up the 'Create New Hyperlink' module, for the end user to choose, either a local URL or any other URL.

```
[ActiveEvent(Name = "Magix.Core.GetImageURL")]
protected void Magix_Core_GetImageURL(object sender, EventArgs e)
```

Level2: Will open up the 'Create New Image' module, for the end user to choose, either a local URL or any other URL for the source of an img HTML element, or something similar

50. Access_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Publishing.Access_Controller

Description:

Level3: Helps out sorting out which Pages and Menu Items which Users and Roles and such have access to

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.VerifyUserHasAccessToPage")]

protected void Magix_Publishing_VerifyUserHasAccessToPage(object sender, ActiveEventArgs e)

Level3: Returns 'STOP' to true unless User has access to Page, either explicitly or implicitly

[ActiveEvent(Name = "Magix.Publishing.FindFirstChildPageUserCanAccess")]

protected void Magix_Publishing_FindFirstChildPageUserCanAccess(object sender, ActiveEventArgs e)

Level3: Will recursively traverse children until it find the first page [breadth first] User has access to from given page

[ActiveEvent(Name = "Magix.Publishing.GetRolesListForPage")]

protected void Magix_Publishing_GetRolesListForPage(object sender, ActiveEventArgs e)

Level3: Will return a List of Roles that have explicit access [or not] to the given Page

[ActiveEvent(Name = "Magix.Publishing.ChangePageAccess")]

protected void Magix_Publishing_ChangePageAccess(object sender, ActiveEventArgs e)

Level3: Will change the Access rights for a specific page

```
[ActiveEvent(Name = "Magix.Publishing.PageObjectDeleted")]  
protected void Magix_Publishing_PageObjectDeleted(object sender, ActiveEventArgs e)
```

Level3: Handled to make sure we delete all WebPageRoleAccess objects belonging to WebPage too

51. QRGenerator_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.QRGenerator.QRGenerator_Controller

Description:

Level2: Implements logic to create, generate and handle QR-Codes

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Publishing.GetDashBoardDesktopPlugins")]

protected void Magix_Publishing_GetDashBoardDesktopPlugins(object sender, ActiveEventArgs e)

Level2: Will populate the Desktop with the Vanity QR Code Generator Icon

[ActiveEvent(Name = "Magix.QCodes.LaunchGenerator")]

protected void Magix_QCodes_LaunchGenerator(object sender, ActiveEventArgs e)

Level2: Will launch the Vanity QR Code Generator. The Vanity QR Code is a QR Code [Quick Response] Code Generator which allows you to create QR Codes pointing to your own URLs, or URLs integral to the system itself

```
[ActiveEvent(Name = "Magix.QRCodes.CreateQRCode")]  
protected void Magix_QRCodes_CreateQRCode(object sender, ActiveEventArgs e)
```

Level2: Will create a QR Code pointing to the given 'URL' parameter. 'Scale' defines how many pixels will be used to render each individual dot in the QR Code, normally 4 is a nice number. 'ErrCor' is the Error Correction level, meaning how much redundancy the code will contain, legal values are L,M,Q,H, meaning Low, Medium, Quality, High. Better quality equals larger and more complex, but also more resilient QR Codes. 'Text', will if given, be rendered at the bottom of the code, underneath, as an explanation. 'RoundedCorners' is an integer value, and will become the radius of the rounded borders, if given. 'AntiPixelate' will, if true, skew the pixels so that it doesn't look as squarish anymore. 'BGImage' will be rendered as a texture as the background instead of where the white normally is. 'FGImage' will be rendered as foreground texture if given. 'FontName' will be used as font for rendering the description 'Text' parts. 'FontSize' expects to be an integer, and will define the size of the text description rendered. 'FontColor' can either be a named color, or an HEX value, such as #9fb430. 'Rotate' defines how many degrees the code will be rotated. Should be an integer value between 0 and 360. 'FileName' defines where you want the QR Code to be saved.

52. Settings_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Settings.Settings_Controller

Description:

Level2: Helper logic to assist with Settings and updating of settings. PS! Unless you have this controller in your Application Pool, Enabled, then direct changes, through the DB Admin module to Settings will NOT affect the Application Pool before it's being re started ...

Methods

[ActiveEvent(Name = "Magix.Core.ApplicationStartup")]
private void Magix_Core_ApplicationStartup(object sender, EventArgs e)

Level2: Need to override 'DBAdmin.Data.ChangeSimplePropertyValue' to make sure settings are 'flushed' if they're being updated

[ActiveEvent(Name = "DBAdmin.Data.ChangeSimplePropertyValue- Override")]
protected void DBAdmin_Data_ChangeSimplePropertyValue_Override(object sender, EventArgs e)

Level2: Overrides 'DBAdmin.Data.ChangeSimplePropertyValue' to make sure settings are 'flushed' if they're being updated. Calls original logic, then resets the settings and reloads them. This because settings are cached on Application Level for speed issues

[ActiveEvent(Name = "Magix.Core.SaveSettingsSection")]
protected void Magix_Core_SaveSettingsSection(object sender, EventArgs e)

Level2: Allows for saving of a list of key/value pairs of settings in one swoop. Settings are in the 'Section' list of nodes, while the name of the Section Group is in the 'SectionName' parameter

[ActiveEvent(Name = "Magix.Core.LoadSettingsSection")]
protected void Magix_Core_LoadSettingsSection(object sender, EventArgs e)

Level2: Returns a 'Section' of settings according to how they were previously saved, if it exists

53. Signature_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.Signature.Signature_Controller

Description:

Level2: Signature Plugin Control for Publishing system to allow for loading of a different plugin module from e.g. the click of a row button or a single-view button etc.

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Signature.LoadSignature")]

protected void Magix_Signature_LoadSignature(object sender, ActiveEventArgs e)

Level2: Will load the Signature module, asking for Signature from the user and saving associated with the record. Will replace the module that raised the Action with the Signature module

54. TalkBack_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.TalkBack.TalkBack_Controller

Description:

Level2: Contains helper logic for the TalkBack module [forums] in Magix

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Talkback.GetPostings")]

protected void Magix_Talkback_GetPostings(object sender, ActiveEventArgs e)

Level3: Will fetch the top 5 postings with the most recent activity within

[ActiveEvent(Name = "Magix.TalkBack.LaunchForum")]

protected void Magix_TalkBack_LaunchForum(object sender, ActiveEventArgs e)

Level2: Will launch the Talkback Module and allow for posting questions and such

[ActiveEvent(Name = "Magix.Talkback.CreatePost")]

protected void Magix_Talkback_CreatePost(object sender, ActiveEventArgs e)

Level3: Will create a new posting with the Header and Body combination linked to the User.Current. If Parent, will put it into the Parent as a Reply

55. ToolTip_Controller

[ActiveController]

Magix.Brix.Components.ActiveControllers.ToolTip.ToolTip_Controller

Description:

Level2: Contains helper logic for displaying and maintaining previous/next Tip of today and such

Basetypes

Magix.Brix.Loader.ActiveController

Methods

[ActiveEvent(Name = "Magix.Core.GetPreviousToolTip")]

private void Magix_Core_GetPreviousToolTip(object sender, EventArgs e)

Level2: Returns the Previous Tooltip on a 'Per User' level. Meaning, it'll keep track of which Tooltip has been shown to which User

[ActiveEvent(Name = "Magix.Core.GetNextToolTip")]

private void Magix_Core_GetNextToolTip(object sender, EventArgs e)

Level2: Returns the Next Tooltip on a 'Per User' level. Meaning, it'll keep track of which Tooltip has been shown to which User

[ActiveEvent(Name = "Magix.Core.GetAllToolTips")]

private void Magix_Core_GetAllToolTips(object sender, EventArgs e)

Level2: Returns all Tooltip instances to caller

56. Clips

[ActiveModule]

Magix.Brix.Components.ActiveModules.ClipBoard.Clips

Description:

Level2: Will display a list of all of your ClipBoard items

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

57. Text

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Text

Description:

Level2: Encapsulates a Text module. Pass in 'Text' and optionally 'Tag'

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

58. Separator

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Separator

Description:

Level2: Encapsulates a [dead] html Horizontal Ruler [hr] element for you if you need one

59. PickColorOrImage

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.PickColorOrImage

Description:

Level2: Colorpicker Module for allowing the end user [or you] to pick colors in the system. Also has support for picking images for use as textures as alternatives to picking color

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.ColorPicker.FileSelected")]

protected void Magix_Core_ColorPicker_FileSelected(object sender, EventArgs e)

Level2: Will raise the SelectEvent with the 'FileName' property set to an image file the end user has just selected

60. ViewPortContainer

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.ViewPortContainer

Description:

Level2: Encapsulates a ViewPort Container Component, basically a container for other modules

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.ClearViewportContainer")]

protected void Magix_Core_ClearViewportContainer(object sender, EventArgs e)

Level2: Clears the Viewport for Modules. Make sure you pass in the correct 'Position', and that it matches the 'Container' parameter in the module's DataSource

[ActiveEvent(Name = "Magix.Core.ContainerAboutToBeCleared")]

protected void Magix_Core_ContainerAboutToBeCleared(object sender, EventArgs e)

Level2: Handled to be able to clear container upon parent container cleared ...

[ActiveEvent(Name = "Magix.Core.GetNumberOfChildrenOfContainer")]

protected void Magix_Core_GetNumberOfChildrenOfContainer(object sender, EventArgs e)

Level2: Will return the number of Active Modules [or controls] a specific Viewport Container contains. Useful for determining if a specific container is available or not

```
[ActiveEvent(Name = "Magix.Core.LoadActiveModule")]  
protected void Magix_Core_LoadActiveModule(object sender, ActiveEventArgs e)
```

Level2: Handled to make it possible to load Active Modules into this Viewport's containers. Make sure the 'Position' parameter matches the 'Container' parameter passed in during loading the ViewPortContainer module

61. Tree

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Tree

Description:

Level2: Shows a Tree module for the end user for him to navigate and select single nodes from. Change its properties by passing in 'TreeCssClass' or 'NoClose'. 'Items' should contain the tree items in a hierarchical fashion with e.g. 'Item/i-1/' containing 'Name' 'CssClass' and 'ToolTip'. 'Name' being minimum. Child items of 'Items/i-1' should be stored in the 'Items/i-1/Items' node. Will raise 'GetItemsEvent' upon needing to refresh for some reasons, and 'ItemSelectedEvent' with 'SelectedItemID' parameter as selected item ID upon user selecting an item

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.UpdateGrids")]

protected void Magix_Core_UpdateGrids(object sender, EventArgs e)

Level2: Overrides to make sure we also update this bugger upon changing of these types of records, but only if 'relevant'

[ActiveEvent(Name = "Magix.Core.SetTreeSelectedID")]

protected void Magix_Core_SetTreeSelectedID(object sender, EventArgs e)

Level2: Sets the selected tree item to the 'ID' given

```
[ActiveEvent(Name = "Magix.Core.ExpandTreeSelectedID")]  
protected void Magix_Core_ExpandTreeSelectedID(object sender, EventArgs e)
```

Level2: Makes sure the 'currently selected tree item' becomes expanded, if any are selected

```
[ActiveEvent(Name = "DBAdmin.Data.ChangeSimplePropertyValue")]  
protected void DBAdmin_Data_ChangeSimplePropertyValue(object sender, EventArgs e)
```

Level3: Overridden to handle some internal events

```
[ActiveEvent(Name = "Magix.Core.GetSelectedTreeItem")]  
protected void Magix_Core_GetSelectedTreeItem(object sender, EventArgs e)
```

Level2: Returns the selected tree item as 'ID' if any is selected

62. TipOfToday

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.TipOfToday

Description:

Level2: Shows one tip of today with the option for the User to browse forward or backward to read more. Raises 'Magix.Core.GetPreviousToolTip' and 'Magix.Core.GetNextToolTip' to get its next and previous tips

Basetypes

Magix.Brix.Loader.IModule

Methods

public void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

63. Header

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Header

Description:

Level2: Encapsulates a header [h1] control for you to create headers for your pages and apps. Load it and raise 'Magix.Core.SetFormCaption' to set the header

Methods

[ActiveEvent(Name = "Magix.Core.SetFormCaption")]

protected void Magix_Core_SetFormCaption(object sender, EventArgs e)

Level2: Sets the caption of the header [h1] control

64. Clickable

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Clickable

Description:

Level2: Encapsulates a Clickable Component, basically a button. Which you can load as a module, and trap clicks to. Set its 'Text', 'ButtonCssClass', 'Enabled' and 'ToolTip' params to customize it till your needs. Will raise 'Event' when clicked

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.EnabledClickable")]

protected void Magix_Core_EnabledClickable(object sender, ActiveEventArgs e)

Level2: Enables or disables [according to 'Enabled'] button

65. AnalogClock

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.AnalogClock

Description:

Level2: Shows an Analog Clock module

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

66. MessageBox

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.MessageBox

Description:

Level2: Implements logic for showing the end user a message box, asking for confirmation or something similar when needed. 'Text' will be the text shown to the user, which he should read and take a stand in regards to. Dropping the 'Cancel' Node sets Cancel to in-visible if you wish. 'OK/Event' will be raise with 'OK' node if OK button is clicked. 'Cancel/Event' will be raised with 'Cancel' node if Cancel button is clicked

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

67. ImageModule

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.ImageModule

Description:

Level2: Control for displaying images in your app. Either clickable or static images. Pass in 'ImageURL', 'AlternateText', 'ChildCssClass' and 'Description' to modify it according to your needs. Description, if given, will add a label underneath the image. Use 'Seed' to have multiple Images on same page and to separate between different instances of them. If 'Events/Click' is given, it'll raise that event upon clicking the image

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.ChangelImage")]

protected void Magix_Core_ChangelImage(object sender, EventArgs e)

Level2: Updates the image to a new 'ImageURL'. But only if 'Seed' is given and correct according to 'Seed' given when loaded

68. ImageList

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.ImageList

Description:

Level2: Modules for displaying a List of Images. Pass in your Image list in the 'Items' parameter to the LoadModule method

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

69. Uploader

[ActiveModule]

Magix.Brix.Components.ActiveModules.CommonModules.Uploader

Description:

Level3: ActiveModule to help create upload files logic. Will raise 'Magix.Uploader.FileUploaded' consecutively for every file uploaded, and 'Magix.Uploader.FileBatchUploadFinished' when all files in one drag and drop operation are done. Alternatively it may raise the event defined in 'FileUploadedEvent' or 'AllFilesUploadedEvent'

Basetypes

Magix.Brix.Loader.ActiveModule

70. ViewClassContents

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewClassContents

Description:

Level2: Will show all objects of type 'FullTypeName' in a Grid from which the user can filter, edit, change values, delete objects, create new objects and such from. Basically the 'show single table' logic of the Magix 'Database Enterprise Manager'. If 'IsCreate', will allow for creation of objects of the 'FullTypeName' by the click of a button

Basetypes

Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

protected override void DataBindDone()

Level4: Called when databinding are done

[ActiveEvent(Name = "Magix.Core.RefreshWindowContent")]

protected override void RefreshWindowContent(object sender, ActiveEventArgs e)

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

Properties

```
protected override System.Web.UI.Control  
Level4: The control being our 'table element'      TableParent
```

71. ViewListOfObjects

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewListOfObjects

Description:

Level2: Basically the same as ViewClassContents, though will only show objects 'belonging to a specific object [ParentID] through a specific property ['PropertyName'] and allow for appending, and not creation of new objects of 'FullTypeName'. Raises 'DBAdmin.Data.GetListFromObject' to get objects to display in Grid. Override the Append button's text property with 'AppendText'. Other properties are as normal from mostly every grid in Magix such as 'IsDelete', 'IsRemove' etc

Basetypes

Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

protected override void DataBindDone()

Level4: Called when databinding are done

Properties

```
protected override System.Web.UI.Control  
Level4: The control being our 'table element' TableParent
```


72. ViewSingleObject

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ViewSingleObject

Description:

Level2: Contains the logic for editing and viewing one single ActiveType object, with all of its properties. Can become initiated in two different states, one of which is 'edit object reference from another object' which will allow for changing and removing the reference, the other is plain old 'edit the thing' mode. Supports 'ChildCssClass' and several other properties. Most of the common properties from the Database Enterprise Manager is included. Will raise 'DBAdmin.Form.ChangeObject' if user attempts to change the reference. Will raise 'DBAdmin.Data.RemoveObject' when object reference is removed. Changing the reference or removing it is only enabled if 'IsChange' and/or 'IsRemove' is given as true

Basetypes

Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

[ActiveEvent(Name = "DBAdmin.Data.DeleteObject- HelperClearer")]

protected void DBAdmin_Data_DeleteObject_HelperClearer(object sender, EventArgs e)

Level2: Simply here to close window upon deletion of instance object

```
[ActiveEvent(Name = "DBAdmin.Form.CheckIfActiveTypesBeingSingleEdited")]  
protected void DBAdmin_Form_CheckIfActiveTypesBeingSingleEdited(object sender,  
ActiveEventArgs e)
```

Level2: Will return 'Yes' == true if the given 'ID' matches the object being edited

```
[ActiveEvent(Name = "DBAdmin.Form.ChangeCssClassOfModule")]  
protected void DBAdmin_Form_ChangeCssClassOfModule(object sender, EventArgs e)
```

Level2: Will change the CSS class of the editing parts of the module if the 'FullTypeName' and the 'ID' matches. Useful for setting CSS class of specific 'Edit Object Module'

73. FindObject

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.FindObject

Description:

Level2: Basically identical to ViewClassContents, with one addition. This module will show a 'filter field' to the end user which the user can use to 'search for items' from within.

Basetypes

Magix.Brix.Components.ListModule

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

protected override void DataBindDone()

Level4: Called when databinding are done

[ActiveEvent(Name = "Magix.Core.RefreshWindowContent")]

protected override void RefreshWindowContent(object sender, ActiveEventArgs e)

Level4: Handled to make sure we re-databind at the right spots. This event is raised whenever a child window is closed, and other windows might need to refresh. We're making sure its either a 'ClientID' match or the incoming 'ClientID' is 'LastWindow' before we do our update here. 'LastWindow' is true if it's the 'last popup window'

Properties

```
protected override System.Web.UI.Control  
Level4: The control being our 'table element' TableParent
```

74. BrowseClasses

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.BrowseClasses

Description:

Level2: Contains UI for letting the end user browse the ActiveType classes within his system as a Database Enterprise Management tool. Allows for seeing all classes in a Tree hierarchy and letting him select classes, which again will trigger editing of records in that class. Kind of like the Database Enterprise Management tool for Magix. Will raise 'DBAdmin.Data.GetClassHierarchy', which you can handle if you have 'meta types' you wish to display

Basetypes

Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

75. ConfigureColumns

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ConfigureColumns

Description:

Level2: Allows for editing of 'visible columns' per type level. Will show a form from which the end user can tag off and on the different columns he wish to see in his grid. Databinds towards 'WhiteListColumns' and raises 'DBAdmin.Data.ChangeVisibilityOfColumn' when visibility of column changes with 'Visible', 'FullTypeName' and 'ColumnName' as parameters

Basetypes

Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

76. ConfigureFilters

[ActiveModule]

Magix.Brix.Components.ActiveModules.DBAdmin.ConfigureFilters

Description:

Level2: Will allow the end user to configure his filters, meaning what filter to filter each column by. There are many different types of filter in Magix. LIKE, Equal and so on. This form allows you to globally set the filters for specific types on specific columns of those types

Basetypes

Magix.Brix.Components.Module

Magix.Brix.Loader.IModule

Methods

public override void InitialLoading(Node node)

Level3: Will be called when the Module is initially loaded with the initializationObject parameter you pass into your LoadModule - if any.

77. ShowClassDetails

[ActiveModule]

Magix.Brix.Components.ActiveModules.Documentation.ShowClassDetails

Description:

Level2: Doxygen helper class for displaying documentation about members of classes and such for our documentation system. Takes in 'FullName', 'Description' and so on. Will create a grid if displaying items according to the structure given

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

78. RichEdit

[ActiveModule]

Magix.Brix.Components.ActiveModules.Editor.RichEdit

Description:

Level2: Contains the UI for the RichEditor or WYSIWYG editor of Magix. That little guy resembling 'word'. Specify a 'SaveEvent' to trap when 'Text' is being edited.

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Brix.Core.GetRichEditorValue")]
protected void Magix_Brix_Core_GetRichEditorValue(object sender, EventArgs e)

Level2: Will flat out return the Value of the RichEditor. Does NOT support multiple editors on the same screen

79. Explorer

[ActiveModule]

Magix.Brix.Components.ActiveModules.FileExplorer.Explorer

Description:

Level2: Containe the UI for the Explorer component, which allows you to browse your File System on your server, through your browser. Basically a File System Explorer kind of control, which allows for renaming, deleting, and editing [to some extent] the files in your installation. Can be instantiated in Select mode by setting its 'IsSelect' input parameter. If 'CanCreateNewCssFile' is true, the end user is allowed to create a new default CSS file which he can later edit. 'RootAccessFolder' is the root of the system from where the current user is allowed to browse, while 'Folder' is his current folder. The control does some basic paging and such, and has support for will raise 'Magix.FileExplorer.GetFilesFromFolder' to get its items. The module will raise the value of the 'SelectEvent' paeameter when an item has been selected. The module supports browsing hierarchical folder structures

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

80. EditAsciiFile

[ActiveModule]

Magix.Brix.Components.ActiveModules.FileExplorer.EditAsciiFile

Description:

Level2: Kind of like Magix' version of 'Notepad'. Allows for editing of textfiles or text fragments, and allow for saving them. Pass in either 'File' [pointing to an absolute file name] or 'Text' which should be textual fragments. If 'NoSave' is True, it won't allow for saving of the document

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

81. Tetris

[ActiveModule]

Magix.Brix.Components.ActiveModules.Games.Tetris

Description:

Level2: Contains the Tetris game

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

82. Slider

[ActiveModule]

Magix.Brix.Components.ActiveModules.Menu.Slider

Description:

Level2: Contains the UI for our SlidingMenu module, used in the administrator dashboard. Takes a recursive 'Items' structure containing 'Caption' and 'Event' which will be raised when clicked ['Event']

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

83. StyleBuilder

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaForms.StyleBuilder

Description:

Level2: Style builder for creating CSS style collections and such for animations, CSS classes, styles etc

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

```
[ActiveEvent(Name = "Magix.MetaForms.FGColorColorWasPickedForBorderDeselect")]  
protected void Magix_MetaForms_FGColorColorWasPickedForBorders(object sender,  
ActiveEventArgs e)
```

Level2: Sets the border-color property of the Widget

```
[ActiveEvent(Name = "Magix.MetaForms.FGColorColorWasPickedDeselect")]  
protected void Magix_MetaForms_FGColorColorWasPicked(object sender, ActiveEventArgs e)
```

Level2: Will change the Foreground Color setting of the builder to the given 'Color' parameter

```
[ActiveEvent(Name = "Magix.MetaForms.BGGradientStartColorWasPicked")]
protected void Magix_MetaForms_BGGradientStartColorWasPicked(object sender,
ActiveEventArgs e)
```

Level2: Will set the Background Gradient Color selection to the given 'Color' parameter value

```
[ActiveEvent(Name = "Magix.MetaForms.BGGradientStopColorWasPickedDeselect")]
protected void Magix_MetaForms_BGGradientStopColorWasPicked(object sender,
ActiveEventArgs e)
```

Level2: Will set the Background Gradient Color selection to the given 'Color' parameter value

```
[ActiveEvent(Name = "Magix.MetaForms.TextShadowColorWasPicked")]
protected void Magix_MetaForms_TextShadowColorWasPicked(object sender, ActiveEventArgs
e)
```

Level2: Will set the Text Shadow Color for the widget to the 'Color' value given

```
[ActiveEvent(Name = "Magix.MetaForms.ShadowColorWasPickedDeselect")]
protected void Magix_MetaForms_ShadowColorWasPicked(object sender, ActiveEventArgs e)
```

Level2: Will set the Shadow Color for the widget to the 'Color' value given

```
[ActiveEvent(Name = "Magix.MetaForms.BGColorColorWasPickedDeselect")]
protected void Magix_MetaForms_BGColorColorWasPicked(object sender, ActiveEventArgs e)
```

Level2: Will change the Background Color setting of the builder to the given 'Color' parameter

```
[ActiveEvent(Name = "Magix.Core.RefreshWindowContent")]
protected void Magix_Core_RefreshWindowContent(object sender, ActiveEventArgs e)
```

Level2: Overridden to set focus back to first element in active MultiViewPanel

84. MetaView_Form

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaForms.MetaView_Form

Description:

Level2: Encapsulates a MetaForm through a PublisherPlugin which you can inject into your sites

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.MetaForms.LoadNewMetaForm")]

protected void Magix_MetaForms_LoadNewMetaForm(object sender, EventArgs e)

Level2: Expects a 'Container' parameter, and if it matches the Container of this specific module, it will change the MetaForm, load that specific MetaForm into itself, and run the Init Actions of that specific MetaForm as if it was just loaded normally

[ActiveEvent(Name = "Magix.MetaView.SetFocusToFirstTextBox")]

protected void Magix_MetaView_SetFocusToFirstTextBox(object sender, EventArgs e)

Level2: Will set Focus to the first TextBox in the form if raised from 'within' the current Meta Form, or explicitly has its 'OriginalWebPartID' overridden to reflect another WebPart ID on the page


```
[ActiveEvent(Name = "Magix.MetaView.EmptyForm")]  
protected void Magix_MetaView_EmptyForm(object sender, EventArgs e)
```

Level2: Will empty the form, and set all of its values back to their defaults, but only the ones with a valid Info field, and which are editable for the end user

```
[ActiveEvent(Name = "Magix.MetaForms.CreateNodeFromMetaForm")]  
protected void Magix_MetaForms_CreateNodeFromMetaForm(object sender, EventArgs e)
```

Level2: Will serialize all Form Widgets with a valid Info field value such that a Node structure is created according to the form widget values

```
[ActiveEvent(Name = "Magix.MetaForms.DataBindForm")]  
protected void Magix_MetaForms_DataBindForm(object sender, EventArgs e)
```

Level2: Will DataBind every Widget of your MetaForm such that every Widget property which starts with 'DataSource' and then any number of square brackets, e.g. DataSource["Objects"][0]["ID"] will then fetch the first Node child of 'Objects' node from your DataSource and databind whatever property you are putting this value into

Properties

```
public string MetaFormName
```

Level2: Which form to load up into the Meta Form View. Must be the unique Name to a Meta Form object

85. EditMetaForm

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaForms.EditMetaForm

Description:

Level2: Helper for editing and creating Meta Form objects

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.MetaForms.RefreshEditableMetaForm")]

protected void Magix_MetaForms_RefreshEditableMetaForm(object sender, ActiveEventArgs e)

Level2: Refreshes and re-databinds the currently edited Meta Form. Useful if you've got wizards and similar types of Action Buttons that modifies the form/widgets somehow, and you need to re-render them with their new settings

[ActiveEvent(Name = "Magix.MetaForms.SetActiveEditingMetaFormWidget")]

protected void Magix_MetaForms_SetActiveEditingMetaFormWidget(object sender, ActiveEventArgs e)

Level2: Sets the currently active editing Widget to the given 'ID'

[ActiveEvent(Name = "Magix.ClipBoard.PasteItem")]

protected void Magix_ClipBoard_PasteItem(object sender, ActiveEventArgs e)

Level2: Handled to make it possible to Paste Widget Hierarchies and similar onto the form

86. MetaView_Single

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaView.MetaView_Single

Description:

Level2: UI parts for showing a MetaView in 'SingleView Mode'. Basically shows a form, with items dependent upon the look of the view. This is a Publisher Plugin module. This form expects to be given a 'MetaViewName', which will serve as the foundation for raising the 'Magix.MetaView.GetViewData' event, whos default implementation will populate the node structure according to the views content in a Key/Value pair kind of relationship. This will serv as the foundation for the module to know which types of controls it needs to load up [TextBoxes, Buttons etc]Handles the 'Magix.MetaView.SerializeSingleViewForm' event, which is the foundation for creating new objects upon clicking Save buttons etc.This is the PublisherPlugin you'd use if you'd like to have the end user being able to create a new MetaObject

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.MetaView.GetWebPartsContainer")]

protected void Magix_MetaView_GetWebPartsContainer(object sender, ActiveEventArgs e)

Level2: Will return the Container's ID back to caller [e.g. "content1"] if it's the correct WebPartTemplate Container according to the requested 'PageObjectTemplateID'

```
[ActiveEvent(Name = "Magix.MetaView.SetFocusToFirstTextBox")]
protected void Magix_MetaView_SetFocusToFirstTextBox(object sender, EventArgs e)
```

Level2: Will set Focus to the first TextBox in the form if raised from 'within' the current MetaView, or explicitly has its 'OriginalWebPartID' overridden to reflect another WebPart ID on the page

```
[ActiveEvent(Name = "Magix.MetaView.SerializeSingleViewForm")]
protected void Magix_MetaView_SerializeSingleViewForm(object sender, EventArgs e)
```

Level2: Will serialize the form into a key/value pair back to the caller. Basically the foundation for this control's ability to create MetaObjects. Create an action, encapsulating this event, instantiate it and raise it [somehow] when user is done, by attaching it to e.g. a Save button, and have the form serialized into a brand new MetaObject of the given TypeName

```
[ActiveEvent(Name = "Magix.MetaView.EmptyForm")]
protected void Magix_Meta_Actions_EmptyForm(object sender, EventArgs e)
```

Level2: Will 'empty' the current form. Useful in combination with Save or Clear button

Properties

```
public string ViewName
```

Level2: The name of the MetaView to use as the foundation for this form

87. MetaView_Multiple

[ActiveModule]

Magix.Brix.Components.ActiveModules.MetaView.MetaView_Multiple

Description:

Level2: UI parts for showing a MetaView in 'MultiView Mode'. Basically shows a grid, with items dependent upon the look of the view. This is a Publisher Plugin module. Basically a completely 'empty' module whos only purpose is to raise the 'Magix.MetaView.ShowMetaViewMultipleInCurrentContainer' event, whos default implementation will simply in its entirety replace this Module, hence not really much to see here. This is the PublisherPlugin you'd use if you'd like to see a 'list of MetaObjects' on the page

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

Properties

public string ViewName

Level2: The name of the MetaView to use as the foundation for this form

88. ImageGallery

[ActiveModule]

Magix.Brix.Components.ActiveModules.PublisherImage.ImageGallery

Description:

Level1: Represents an Image Gallery plugin for the Publishing system. Allows for browsing of Images through a Gallery lookalike UI

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

Properties

public string GalleryName

Level1: The Unique name of the Image Gallery

89. Uploader

[ActiveModule]

Magix.Brix.Components.ActiveModules.PublisherUploader.Uploader

Description:

Level2: File Uploader plugin module for the Publishing system. By having this module on your WebPage, you will allow the end user to drag and drop files onto anywhere on the browser, which will trigger uploading of those files to the server

Basetypes

Magix.Brix.Loader.ActiveModule

Properties

public string Filter

Level2: If given, will only show files of given types. Should contain a string of file filters separated by semi-colons. E.g. "*.png;*.jpg;" will allow for all files having either the .png or the .jpg extensions. By default the control tolerates image files, CSV files, PDF files, CSS files, text files [.txt],

public string ActionName

Level2: If given, will Raise the given Action with the 'FileName', full relative path, after saving file on server. The default value of 'NO-ACTION' will make sure no Actions are being raised. If you wish you can separate multiple actions with the pipe symbol (|) to have multiple Actions raised

public string Folder

Level2: What folder on your server you wish the plugin to use for saving file within

90. Header

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.Header

Description:

Level2: A 'header' PublisherPlugin. Basically just an HTML h1 element

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Publishing.ShouldReloadWebPart")]

protected void Magix_Publishing_ShouldReloadWebPart(object sender, EventArgs e)

Level2: Will return false if this webpart can just be updated by asking for new data

Properties

public string Caption

Level2: Basically the content of the h1 element. Default value is 'The Caption of your Header'

91. EditSpecificTemplate

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.EditSpecificTemplate

Description:

Level2: Allows for editing of WebPageTemplate objects. Contains most of the UI which you're probably daily using while adding and creating new templates and such

Basetypes

Magix.Brix.Loader.ActiveModule

92. LogInOutUser

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.LogInOutUser

Description:

Level2: PublisherPlugin containing most of the UI for allowing a user to login/out of your website. Can be set into both OpenID mode and 'only native mode' or both. Raises 'Magix.Publishing.GetStateForLoginControl' to determine the state of the module, meaning if it should show both OpenID logic and native logic or only one of them. Will raise 'Magix.Core.UserLoggedOut' if user logs out and 'Magix.Core.LogInUser' if the user tries to log in. The 'Magix.Core.LogInUser' default implementation again will raise 'Magix.Core.UserLoggedIn' if it succeeds. It'll pass in 'OpenID' if user has chosen to log in with OpenID and 'Username'/'Password' if user choses to login natively

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

[ActiveEvent(Name = "Magix.Publishing.ShouldReloadWebPart")]
protected void Magix_Publishing_ShouldReloadWebPart(object sender, EventArgs e)

Level2: Will return false if this webpart can just be 'reused' to the next page

[ActiveEvent(Name = "Magix.Publishing.GetTemplateColumnSelectTypeOfLoginControl")]
protected void Magix_Publishing_GetTemplateColumnSelectTypeOfLoginControl(object sender, EventArgs e)

Level3: Implementation of 'Get Select Type Of Login Control' for Magix. Will return a Select Lst back to caller

Properties

`public string LoginMode`

Level2: Which mode you wish to use for your login control. Legal values are 'OpenID', 'Native' and 'Both'. Signifying the obvious, both being the default which will allow you to log in either with username/password combination or an OpenID claim

93. TopMenu

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.TopMenu

Description:

Level2: PublisherPlugin containing a conventional 'File Menu' type of menu which normally is expected to find at the top of a page, which will 'drop down' child selection boxes for being able to select children. Useful for conventional applications, which should look like legacy code, or something. Takes the exact same input parameters as the SliderMenu PublisherPlugin

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Publishing.ShouldReloadWebPart")]

protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)

Level2: Will return false if this webpart can just be 'reused' to the next page

94. SliderMenu

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.SliderMenu

Description:

Level2: PublisherPlugin containing the UI for the Sliding Menu Publisher Plugin. Meaning the default menu to the left in the front-web parts, which you can choose to inject into a WebPartTemplate in one of your Templates if you wish. Basically just loads the items once through raising the 'Magix.Publishing.GetSliderMenuItems' event, which should return the items as a 'Items' list, containing 'Caption', 'Event', 'Event' [Event name] and 'Event/WebPageURL' which normally will contain the page's URL

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Publishing.ShouldReloadWebPart")]

protected void Magix_Publishing_ShouldReloadWebPart(object sender, ActiveEventArgs e)

Level2: Will return false if this webpart can just be 'reused' to the next page

95. Content

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.Content

Description:

Level2: A 'content' PublisherPlugin. Basically just a text fragment, that'll be edited through the Magix' RichText or WYSIWYG Editor

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Publishing.ShouldReloadWebPart")]

protected void Magix_Publishing_ShouldReloadWebPart(object sender, EventArgs e)

Level2: Overridden to just 'fetch the text', since such things are easier and faster for the server and such than reloading the entire page. Will basically determine if the container and module name are the same, if they are, it'll return STOP which will stop the reloading of the new module. Then this module will raise 'Magix.Publishing.GetWebPartSettingValue' for the new 'WebPartID', which will return the new Text as 'Value'

Properties

```
public string Text
Level2: The actual content of the Content control ... ;)
```


96. ChildExcerpt

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.ChildExcerpt

Description:

Level2: Will show an excerpt of all its children, sorted with newest first, showing a maximum of PagesCount items. Kind of like a front page to a blog or a news website or something

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

Properties

public int PagesCount

Level2: How many of the latest pages will be shown in the excerpt

97. CreateImage

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.CreateImage

Description:

Level2: Encapsulates a module for selecting an image

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.SelectImage")]

protected void Magix_Core_SelectImageForHyperLink(object sender, EventArgs e)

Level2: Sink fallback for allowing the user to select to link to any object within his filestructure

98. EditSpecificPage

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.EditSpecificPage

Description:

Level2: Allows for editing of one single specific WebPage in the system. Contains most of the UI which you're probably daily using while adding and creating new pages and such

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Publishing.PageWasDeleted")]

protected void Magix_Publishing_PageWasDeleted(object sender, EventArgs e)

Level2: Needs to be handled here, in case it was 'this page'

[ActiveEvent(Name = "Magix.Publishing.PageWasUpdated")]

protected void Magix_Publishing_PageWasUpdated(object sender, EventArgs e)

Page was somehow updated, and we need to reload the currently editing page

99. CreateLink

[ActiveModule]

Magix.Brix.Components.ActiveModules.Publishing.CreateLink

Description:

Level2: Encapsulates a module for selecting object, image, etc to link to

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

[ActiveEvent(Name = "Magix.Core.SelectImageForHyperLink")]

protected void Magix_Core_SelectImageForHyperLink(object sender, ActiveEventArgs e)

Level2: Sink fallback for allowing the user to select to link to any object within his filestructure

100. Generator

[ActiveModule]

Magix.Brix.Components.ActiveModules.QRGenerator.Generator

Description:

Level2: Vanity QR Code Generator for creating 'beautiful' QR Codes. Also contains an algorithm [burn] which creates more reciliant QR codes that scans faster than conventional square dots

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

101. Sign

[ActiveModule]

Magix.Brix.Components.ActiveModules.Signature.Sign

Description:

Level2: A Signature module, basically a place where the end user can 'sign his name' to confirm a transaction of some sort. Will load up a big white thing, which can be 'drawn upon', together with two buttons, OK and Cancel which will raise the 'CancelEvent' and the 'OKEvent'. 'OKEvent' will pass in 'Signature' being the coords for all the splines that comprises the Signature, which can be stored and later used as input to this Module, which will again load those splines in 'read-only mode'

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

102. Forum

[ActiveModule]

Magix.Brix.Components.ActiveModules.TalkBack.Forum

Description:

Level2: Basically a 'Forum module' which allows for posting and reading and replying to other people's opinions about 'whatever'. Not very good on its own, please use through TalkBack Controller to save head aches

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

103. Login

[ActiveModule]

Magix.Brix.Components.ActiveModules.Users.Login

Description:

Shows a login box with username/password/openid combo. Username/password has preference, but if only OpenID is given, the system will attempt at login you in using the OpenID claim. Note, this is not the Publisher Plugin login box. This is the 'main system one', which you get by going to ?dashboard=true

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

public override void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

104. Settings

Magix.Brix.Components.ActiveTypes.Settings

Description:

Level2: Wrapper class for common/global settings within a Magix-Brix application. Use the overloaded this operator to access Settings through code

Methods

public void Reload()

Level3: Will reload all settings from DB ...

public void Clear()

Level4: Will delete all settings. Notice that this operation cannot be undone!

public void Remove(string key)

Level3: Removes the specific Setting Key form the collection

Properties

public Settings Instance

Level3: Retrieves the singleton instance of your settings. Reference the Settings ActiveTypes in your project, use the Instance property from below, and use the index operator of the class or Get/Set to access specific settings

```
public string this[string key]
```

Level3: Returns or sets the key value as a string

```
public IEnumerable< string > Keys
```

Level4: Returns all the keys which exists.

```
public int Count
```

Level3: Returns the number of settings in total in database

105. TipOfToday

Magix.Brix.Components.ActiveTypes.TipOfToday

Description:

Level3: Helper class for Tip [Today's Tips] logic

Methods

public void CreateTip(string tip)

Level3: Only way to create a tip form the outside

public string Previous(string seed)

Level3: Will return the 'previous' tip with the given seed

public string Next(string seed)

Level3: Will return the 'next' tip with the given seed

public string Current(string seed)

Level3: Will return the 'current' tip with the given seed, and not in any ways increament or decrement the 'currently viewed tips' like the sibling methods will

Properties

public TipOfToday Instance

Level3: Getter to gain access to today's tips

```
public int Count
Level3: Number of tips in database
```

106. Cookie

[ActiveType]

Magix.Brix.Components.ActiveTypes.Distributor.Cookie

Description:

Level2: Encapsulates a Cookie for a specific Domain. Used to be capable of transferring and storing state from one server to another when for instance remoting events are being raised

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

[ActiveField()]

public string Domain

Level2: The Application Path the Cookie was created for, e.g. <http://magixilluminate.com/>. Must end with a /. Magix gives you additional granularity using this class, than the HTTP Cookie parts of the standard gives you by making you capable of separating different cookies according to different paths with a server. This is ofc ourse easy to override, and set back to its original logic. However, especially since Magix are for all practical concerns a single page web app framework [more or less]. This makes sense for Magix as a default behavior. Notice that there are no changes to the standard or Magix' usage of this standard, only the granularity of how Magix internally saves and retrieves cookies for specific applications within the same domain. Change the Setting called 'Magix.Core.MultiplAppsPerDomain' to reset this such that Magix will send its cookies according to how 'it's supposed to do it' according to the HTTP standard. Notice how this makes sense for Magix, since by using Magix you can basically 100% transparently have hundreds of hundreds of virtually overridden events, penetrating the network at all times, going back and forth between hundreds of applications, which again might call back into the original server, though to a different app path

[ActiveField()]

public string Name

Level2: The name of the cookie

[ActiveField()]

public string Value

Level2: The actual value of the cookie. Anything that can be described as a string can in theory be stored here.

[ActiveField()]

public string ActualDomain

Level2: Not really relevant for us, since we're storing cookies on 'application level' anyway, but we must be able to reproduce the Cookie exactly as it came back from the end-point, and hence need to store this value somehow

[ActiveField()]

public DateTime Expires

Level2: When the cookie expires. After this date, it will no longer be attached, but deleted

[ActiveField()]

public DateTime Created

Level2: When the cookie was created

107. Access

[ActiveType]

Magix.Brix.Components.ActiveTypes.Distributor.Access

Description:

Level2: Encapsulates access right for one single other server in regards to whether or not it can legally raise events, and which events it has requested to raise

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

[ActiveField()]

public string SecretKey

Level2: Secret Key which basically is the communication point up to the end point server to supply. Keep this one secret, since it basically authenticates you to that other specific server, and hence encapsulates your access rights on the other party's behalf. Loosing this key, might not only be seen as stupid, but also directly irresponsible on behalf of the server to which this key authenticates you towards

[ActiveField()]

public string UriReferrer

Level2: Not to be trusted, but really just a friendly name for the app. Magix will automatically use the base URL for where it's installed as the referrer

[ActiveField()]

public string IP

Level2: IP address of requesting party

[ActiveField()]

public DateTime Created

Level2: When request was initially initiated

[ActiveField()]

public bool Ignored

Level2: If true, will be ignored every time other side tries to initiate contact

[ActiveField()]

public LazyList< Event > Events

Level2: List of events the other party can legally raise on our server

108. Gallery

[ActiveType]

Magix.Brix.Components.ActiveTypes.Gallery.Gallery

Description:

Level3: Tutorial: Contains the Entity Object or Active Type as we call them for our Gallery Plugin. Please notice that the Gallery plugin is more here intended as an example of how to create your own C# plugins than being an actually useful plugin. Especially considering you are literally way more powerful yourself using the Meta Form Builder and such for creating your own types of galleries. Though it serves as a useful example still for how to create your own C# plugins, if the back-end webparts just are not powerful enough for some tasks you need to solve

Basetypes

Magix.Brix.Data.ActiveType-g

Constructors

```
public      Gallery()
```

Level3: The CTOR is needed since otherwise we might get null reference exceptions since otherwise our default value of our Images will be null

Methods

```
public override void Save()
```

Level3: Should normally contain your domain logic validation logic for your entity types, well known types, domain types, active types - or whatever you want to call them. Anyway, this is where you would do most of your pre-serialization validation

Properties

[ActiveField()]

public string Name

Level3: The friendly name for our gallery, defaults to the username of the person who created it

[ActiveField()]

public string Folder

Level3: Which folder on disc, relative from app main path, where all the images in this same gallery exists

[ActiveField()]

public DateTime Created

Level3: Typical useful property, you would probably want to add to most of your active types too. Helps us sort on relevance. 98% of the time, new is more relevant than old ...

[ActiveField(IsOwner=false)]

public User User

Level3: The User who created the Gallery, if any. Might be null

[ActiveField()]

public LazyList< Image > Images

Level3: The list of Image objects, which again encapsulates the path

109. LogItem

[ActiveType]

Magix.Brix.Components.ActiveTypes.Logging.LogItem

Description:

Class encapsulating log items in Magix. Magix has automated logging components which will log certain aspects about your system and its behavior. These log items are stored in your database through using this active type

Basetypes

Magix.Brix.Data.ActiveType-g

Properties

[ActiveField()]

public DateTime When

Automatically generated date of when log item was created

[ActiveField()]

public string LogItemType

A friendly 'type string' meant to easily allow filtering different categories of log items according to how the end user wants it

[ActiveField()]

public string Header

Friendly [and SHORT!] description of log item

[ActiveField()]

public string Message

More thorough description if feasible

[ActiveField()]

public int ObjectID

And integer value pointing to the ID of the 'relevant object', whatever that is

[ActiveField()]

public int ParentID

Similar to above. Just 'another container' for similar type of data

[ActiveField()]

public string StackTrace

StackTrace, if we're in an exception

[ActiveField()]

public string IPAddress

IP address of user triggering log item. Saved for security reasons

[ActiveField(IsOwner=false)]

public UserBase User

Logged in user triggering log item

[ActiveField()]

public string UserID

ID of cookie on disc for user, if any. Every request to Magix will automatically create a 'tracking cookie', upon from which the user will be automatically associated with, till he either uses a different client, or empties his cache somehow. This is the 'ID' of that tracking cookie. Which makes it possible to 'identify' users, even though they're not logged in

110. MetaForm

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaForms.MetaForm

Description:

Level3: Encapsulates the logic for serializing and storing Meta Forms objects

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Methods

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

public MetaForm Clone()

Will return a deep copy of the entire Meta Object, with all its Child Objects being cloned, and all its properties being cloned. WARNING; Might take insane amounts of time, if your object graph is huge

Properties

[ActiveField()]

public string Name

Level3: Name of Meta Form

[ActiveField()]

public DateTime Created

Level3: Automatically tracked date for when the Meta Form was created

[ActiveField()]

public Node Form

Level3: Serialized Nodes, defining the Control Hierarchy, within this Meta Form. The root Node is the Form Object

public Magix.Brix.Types.Node FormNode

Level3: Will return the Magix.Brix.Types.Node converted structure, and cache it for future requests

111. MetaObject

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaTypes.MetaObject

Description:

The storage for the 'Meta Application System' in Magix. Every time you create a new object, using a Meta View or something similar, a MetaObject is being created to hold this data. A Meta Object is basically an object 'without structure', where you can, by editing your views, maintain the structure you wish for your data. In such a way you can create your own Meta Applications, using Magix, without being imposed onto an existing data-structure in any ways

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Methods

public override void Save()

Making sure every child has this as its parent to avoid 'cache bugs'. Plus other types of logic

public MetaObject CloneAndSave()

Level3: Will return a deep copy of the entire Meta Object, with all its Child Objects being cloned, and all its properties being cloned. WARNING; Might take insane amounts of time, if your object graph is huge. WARNING; This method, will due to the layout graph of the Children property actually SAVE the MetaObject before it returns it, meaning; you don't need to call save yourself

Properties

[ActiveField()]

public string TypeName

The 'type' of your MetaObject. This property is being extensively used through out the entire Magix to separate your MetaObjects into specific 'types' for you. By filtering upon these types, different types of objects can emerge, such as 'Customer' and 'Contacts' and such. Though to play well with others, you should probably prefix all your TypeNames somehow, with e.g. your Company Name or something, to avoid clashing with other modules and other people's/company's MetaObjects

[ActiveField()]

public string Reference

Normally an automatically generated reference of how the object came into existence. E.g. the name of the View that created it, etc

[ActiveField()]

public DateTime Created

An automatically maintained value signifying when the object was created

[ActiveField()]

public LazyList<Property> Values

All 'values', 'properties' or 'fields' of the object

[ActiveField(IsOwner=false)]

public LazyList<MetaObject> Children

The child MetaObjects of this object. Every MetaObject can have as many children as it sees fit. You can this way have one Customer having several Contacts for instance. Later we will create possibilities within the MetaView system to react upon these, intelligently, somehow, and allow the user to 'nest forms' and such. But currently this is unresolved unfortunately. Please notice that these are 'loosely coupled objects', meaning they won't be deleted when parent is deleted, and they can belong to more than one object

```
[ActiveField(RelationName="Children", BelongsTo=true)]  
public MetaObject ParentMetaObject  
The parent of the this object, if any
```

112. Action

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaTypes.Action

Description:

Contains the serialized content of the Meta Action system in Magix. One Action is a wrapper around an Active Event, as described in the O2 Architecture Document. Basically, an Action can be thought of as a 'serialized version' of an ActiveEvent being raised, which you then can choose to raise whenever you wish, at the trigger of a button for instance. By creating your own actions, you can effectively implement your own type of logic and flow in Magix without having to code. Every Action must have a unique name, and is often either referenced by its Name or its ID property.

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Methods

public override void Save()

Overridden to make sure name is unique, among other things

public Action Copy()

Will perform a deep copy of the Action, with all of its parameters, and return back to caller

Properties

[ActiveField()]

public string Name

Level2: Must be unique. Used to de-reference the action in other places. Normal coding convention should be to add at least your company name as the action's prefix or something, to avoid name clashing

[ActiveField()]

public string Description

Level2: Friendly description, shown to user, which should explain the action in 'human readable text' for the end user

[ActiveField()]

public string EventName

Level2: The Name of the Active Event being raised by this action

[ActiveField()]

public string Overrides

Level2: The Name of the Active Event this Action will Override. If this value ends with a '!' then that means that the action will override the existing event handlers, such that the original handlers cannot be called, unless you explicitly somehow makes sure that Overrides + '-Action-Overridden' somehow is raised. If it does not contain a '!' at the end, it will not leave these existing event handlers in oblivion, but also make sure these are called, though after the EventName has been raised. If your Overrides property value contains only '!', the override is considered to be a NULL event handler, meaning it will handle every single active event in your system, meaning unless you raise the inner events yourself, your entire system falls apart. Useful for logging and especially handling some events, just remember to forward the original event, since otherwise it will never be raised, unless choking the event is your intention. Though do NOT choke ALL events, this will crash your system. If this has occurred, you need to manually enter your database and change the value

[ActiveField()]

public LazyList< ActionParams > Params

The parameters to the ActiveEvent. Will become serialized as Nodes being passed into the Active Event

[ActiveField()]

public bool StripInput

If true, will CUT the incoming node, which might be useful to make sure your nodes doesn't grow into 'Kingdom Come'

[ActiveField()]

public DateTime Created

Automatically maintained by Magix to give you a date of creation for the record

113. MetaView

[ActiveType]

Magix.Brix.Components.ActiveTypes.MetaViews.MetaView

Description:

Contains the logic for our views in our 'Meta Application System'. A MetaView is something that defines what the user is supposed to see in regards to MetaObjects. Basically the UI 'representation' of your object graphs and properties. A MetaView can become the 'engine' for a WebPart through the MetaView_Single and MetaView_Multiple plugins for Magix

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Overridden to make sure Name is unique, among other things

public MetaView Copy()

Will perform a deep-copy on the entire MetaView with all its properties and return to caller

Properties

[ActiveField()]

public string Name

Name of MetaView

[ActiveField()]

public string TypeName

The 'Type Name' of the MetaObject associated with this view. If this is a SingleView for instance, it'll create exclusively types of the given TypeName. If it's a MultiView, it'll exclusively show items of this type. Take care when designing your MetaType hierarchy

[ActiveField()]

public LazyList< MetaViewProperty > Properties

All the properties for the MetaView

[ActiveField()]

public DateTime Created

Automatically maintained by Magix to give you a date of creation for the record

114. User

[ActiveType(TableName="docMagix.Brix.Components.ActiveTypes.Users.UserBase")]

Magix.Brix.Components.ActiveTypes.Publishing.User

Description:

Level3: Encapsulates one user in Magix' Publishing system. Inherited to add more, specific to the publishing system, to the class. PS! This class is a fine example of inheriting ActiveTypes for the curious C# developer out there. Basically you've got to override the TableName to point your inherited type to the correct typename, in addition to overriding a "mapped type" in your web.config

Methods

public override void Save()

Overridden to make sure we've got a default AvatarURL before calling base

public new User SelectByID(int id)

Returns the object with the given ID from your data storage.

public new IEnumerable(User) SelectByIDs(params int[])

Returns a list of objects with the given ID from your data storage.

public new User SelectFirst(params Criteria[])

Returns the first object from your data storage which are true for the given criterias. Pass null if no criterias are needed.

public new IEnumerable(User) Select(params Criteria[])

Returns all objects from your data storage that matches the given criterias. Pass null if you want all objects regardless of any values or criterias.


```
public new int CountWhere(params Criteria[])
```

Returns the number of objects in your data storage which is of type User where the given criterias are true.

Properties

[ActiveField()]

```
public string FullName
```

Full name of person

[ActiveField()]

```
public string AvatarURL
```

URL, can either be relative [media/images/myImage.jpg] or absolute [http://g.com/x.jpg]

[ActiveField()]

```
public string Phone
```

Phone number to person

[ActiveField()]

```
public string Address
```

Street Address to person

[ActiveField()]

```
public string City
```

City where person lives

[ActiveField()]

```
public string Zip
```

Zip, yup ...

[ActiveField()]
public string State
State [if US. Ignore if other countries]

[ActiveField()]
public string Twitter
The Twitter Handle to the person, e.g. 'WinergyInc'

[ActiveField()]
public string Facebook
The Facebook username to the person, e.g. 'polterguy'

[ActiveField()]
public DateTime BirthDate
Date of birth

[ActiveField()]
public string Country
Country where person lives

[ActiveField()]
public string Gender
Yup, normally 'Male' or 'Female'

[ActiveField()]
public string Language
Come estas? Comprende ...?

[ActiveField()]
public string Nickname
A small little personal 'nickname', e.g. 'polterguy'

[ActiveField()]
public string TimeZone
Which timezone is person within

[ActiveField()]
public LazyList< OpenIDToken > OpenIDTokens
List of OpenID Claims associated with person

public new int Count
Returns the number of objects in your data storage which is of type T.

115. WebPageTemplate

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPageTemplate

Description:

Serves as a 'recipe' for a WebPage. Every WebPage is built from one of these. Contains the definition of which module to instantiate for instance, but not which parameters to send into it. Also contains the positioning of the WebParts and other common features, such as CSS classes and such. A Template is just that, a 'template' for your WebPages

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Delete()

Level3: Deletes the object from your data storage.

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

[ActiveField()]

public string Name

Friendly name of template. Serves no logical meaning. It is probably smart though to create a coding convention of some sort here

[ActiveField()]

public LazyList< WebPartTemplate > Containers

Our 'WebPart-recipes'. Contains the recipe for every webpart that should be added to our page. The number of WebPartTemplates, and their settings, will define how the WebPage's WebParts will look like. This property 'controls' how the WebParts property of the associated WebPage is being built

[ActiveField()]

public DateTime Created

Automatically kept track of. Keeps the 'created date' of the object

116. WebPart

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPart

Description:

Is one instance of a container on a WebPage. Baically one 'module' if you wish. A WebPage is created of several WebParts, each WebPart again is built from a corresponding WebPartTemplate associated with the WebPageTemplate the page itself is built upon. This class encapsulates the WebPart of this relationship, and hence serves as a wrapper for WebPart settings basically on a per page level

Basetypes

Magix.Brix.Data.ActiveType-g

Properties

[ActiveField(IsOwner=false)]

public WebPartTemplate Container

Which template [recipe] the WebPart is built upon

[ActiveField(BelongsTo=true)]

public WebPage WebPage

Which web page the WebPart belongs to

[ActiveField()]

public LazyList< WebPartSetting > Settings

Settings for this WebPart. Mostly contains 'values' for the ModuleSettingAttribute attributes of the plugin

117. WebPartTemplate

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPartTemplate

Description:

Serves as a 'recipe' for WebParts. Contains stuff such as positioning of webpart, and name of module to inject. Every WebPageTemplate has a list of these guys, which serves as recipe for how the WebPage's WebParts should be built, and which Plugins to load up

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Delete()

Level3: Deletes the object from your data storage.

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

Properties

[ActiveField()]

public string Name

Name of template. Serves no logical purpose. But please make meaningful names

[ActiveField()]

public string ViewportContainer

Which container in our ViewPort are we supposed to get stuffed into. Normally this value is being 'normalized' though, meaning the WebParts are stuffed out in chronological order, according to how they appear in the collection, up until we've either stuffed them all out, or ran out of Viewport Containers

[ActiveField()]

public string ModuleName

Contains the fully qualified name of the module to instantiate. Warning, if modifying this one 'by hand', bad things can happen! This should contain the full name of the Module's C# Class, e.g.; 'MyCompany.MyComponent.MyModule'. These modules must be attributed as PublisherPlugins

[ActiveField()]

public string CssClass

Additional CSS class(es) to be injected into the webpart

[ActiveField(BelongsTo=true)]

public WebPageTemplate PageTemplate

Which page template our webpart template belongs to

[ActiveField()]

public int Width

Width in multiples of 30/70/110 [+40] units up til 24

[ActiveField()]

public int Height

Height in multiples of 18

[ActiveField()]

public int MarginLeft

Margin-Left in multiples of 18. Can go negative to -23

[ActiveField()]
public int MarginTop
Margin-Top in multiples of 18. Can go negative to -30

[ActiveField()]
public int MarginRight
'Padding' in multiples of 30/70/110 [+40] units up til 24. Can go negative to -23

[ActiveField()]
public int MarginBottom
Bottom-Top in multiples of 18. Can go negative to -30

[ActiveField()]
public bool Last
If true, will eliminate all margins [10px spacing] to the right of the webpart

[ActiveField()]
public bool Overflow
If true, will allow the element to overflow in the vertical direction. Meaning it will 'ignore' its height, and just 'flow as high as it is'

118. OpenIDToken

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.OpenIDToken

Description:

Encapsulates an OpenID Token. Meaning an OpenID Claim. Magix has support for serving as both an OpenID Relying Party and an OpenID Provider. This class serializes the OpenID Claims for users when using Magix as an OpenID Relying Party, and associating these claims with specific users. This means you can associate an OpenID claim to your user and then use the OpenID Token to log into Magix later

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Overridden to make sure all Claims are unique within Magix

Properties

[ActiveField()]

public string Name

OpenID Claim or Token. E.g.; myGoogleUsername.blogspot.com

[ActiveField(BelongsTo=true)]

public User User

Which user the claim belongs to, if any

119. WebPage

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPage

Description:

Represents one web page in our system, or one unique URL if you wish. Contains a list of WebParts which again are areas of the screen within your web page. Every page must have at least one webpart, but can have many more. This class encapsulates the logic of 'Pages' in Magix, which is probably easy to understand within the context of publishing or CMS systems. Anyway, a page in Magix might also be a container for your application, plugins and WebParts. WebParts and Plugins again can be either your own creations through MetaViews or something similar, or actual C# code written plugins for your system. One page is often easy to understand if you can perceive it as 'one URL', while it is still much more powerful than any 'CMS Pages' or 'Publishing Pages' out there. Every page is built upon a 'recipe' which is the WebPageTemplate class. The WebPageTemplate class contains the logic for which plugin type it should use etc, while the WebPage contains the settings for the type of plugins instantiated upon opening it

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Methods

public override void Save()

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

public override void Delete()

Level3: Deletes the object from your data storage.

public new WebPage SelectFirst(params Criteria[])

Returns the first action from your data storage which are true for the given criterias. Pass nothing () if no criterias are needed.

Properties

[ActiveField()]

public string Name

Name of your Web Page. Serves nothing but as a friendly name and have no real meaning in the system. Doesn't need to be in any ways unique. Should be 'descriptive' such as if you've got a 'Header' type of WebPart, it might be smart to Name your webpart 'Header' too, or 'Sub-Section-Header' or something similar

[ActiveField()]

public string URL

Needs to be unique system wide, but will do its best at staying unique. Also serves as a Materialized Path for our Page Hierarchy. Please notice though that if you change the URL after the page is created, it'll update the URL's of ALL child pages, and their children again, and so on infinitely inwards. This makes changing the URL after creating [many] children for it a potential very expensive operation

[ActiveField()]

public DateTime Created

Automatically kept track of. Keeps the 'created date' of the WebPage

[ActiveField(IsOwner=false)]

public WebPageTemplate Template

The Template the Page is built upon

[ActiveField()]

public LazyList< WebPart > WebParts

The parts, or containers of the Web Page

[ActiveField()]

public LazyList< WebPage > Children

Children Page objects. Every WebPage might have a list of children within it

[ActiveField(BelongsTo=true)]

public WebPage Parent

Parent web page. Is null if this is the top most page

120. WebPageRoleAccess

[ActiveType]

Magix.Brix.Components.ActiveTypes.Publishing.WebPageRoleAccess

Description:

Gives access to a page according to a specific role. This is the core class in Magix in regards to authorizing access to specific pages. Access in Magix is associated with a User, his Role and the WebPage the user tries to access. You cannot authorize on a more fine-grained level out than WebPages, out of the box. This class encapsulates the authorization logic in Magix. By default everyone have access to everything. If one Role is explicitly granted access, then all other roles will be denied access, unless they too are granted access. In addition access is 'inherited' from the 'Mother Page'. Meaning a child page will be default have the same access rights as its Mother page, unless the child page itself starts granting and denying access to specific pages

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Properties

[ActiveField(IsOwner=false)]

public Role Role

Role to grant users belonging to that role access to the Page in this object. All other roles, which are not having their own access objects towards the page, will now no longer have access to this page

[ActiveField(IsOwner=false)]

public WebPage Page

Page to grant users belonging to role access to. All other roles, which are not having their own access objects towards the page, will no longer have access to this page

121. Posting

[ActiveType]

Magix.Brix.Components.ActiveTypes.TalkBack.Posting

Description:

Level2: One Talkback posting. If Parent is null, this is a top-level posting, and the Children collection might have content. If not, it's a child itself of another top level posting, as in a 'reply'

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Overridden to set the User and When properties

Properties

[ActiveField()]

public string Header

Level2: The descriptive header of the posting

[ActiveField()]

public string Content

Level2: The actual content of the posting

[ActiveField()]
public DateTime When
Level2: Automatically changed value of when posting was created

[ActiveField(IsOwner=false)]
public UserBase User
Level2: Which user created the posting

[ActiveField()]
public LazyList< Posting > Children
Level2: Only top-level postings have children. These children can be perceived as 'replies' to the 'OP' [Original Poster]

[ActiveField(BelongsTo=true)]
public Posting Parent
Level2: If this is null, this is a top level posting, in which case it might have children

122. UserSettings

[ActiveType]

Magix.Brix.Components.ActiveTypes.Users.UserSettings

Description:

Level3: Settings stored on a 'per user level'. Useful for storing simple fact on a per user level

Basetypes

Magix.Brix.Data.ActiveType-g

Properties

[ActiveField()]

public string Name

Level3: The Name of the settings. Must be unique within the user. Normally you should prefix these kinds of 'global storages' with your company name or something, to avoid nameclashings

[ActiveField()]

public string Value

Level3: The Value, stored as string. Convert as you wish yourself

123. UserBase

[ActiveType(TableName="docMagix.Brix.Components.ActiveTypes.Users.UserBase")]

Magix.Brix.Components.ActiveTypes.Users.UserBase

Description:

Level3: A User is a single person using your website, being registered with a username and a password, and hence no longer 'anonymous'. A user is normally recognized through his 'Username', and he has a password which he can use to log into the system, normally. This class encapsulates that logic. PS! This class supports inheriting [though it's not really entirely stable quite yet ...!]

Basetypes

Magix.Brix.Components.ActiveTypeCached-g

Methods

public void RemoveSetting(string name)

Level3: Removes the given setting for the user

public override void Save()

Level3: Overridden to make sure username is unique etc.

public void SaveNoVerification()

Level3: Will save the user 'raw' running none of the verification logic to make sure the data is not violating some logic or something

public override void Delete()

Level3: Overridden to make sure we log this

```
public bool InRole(string roleName)
```

Level3: Returns true if user belongs to a role with the given name [case-in-sensitive search]

```
protected void SaveImpl(bool verify)
```

Level3: Implementation of our two other Save methods

Properties

```
[ActiveField()]
```

```
public string Username
```

Level3: Username. Normally used to login to the system with, in combination with the user's password

```
[ActiveField()]
```

```
public string Password
```

Level3: Password. Unique word, character combinations or a sentence which only the user knows about himself, and which is used to authenticate the user upon logging in

```
[ActiveField()]
```

```
public string Email
```

Level3: Email address of user

```
[ActiveField(IsOwner=false)]
```

```
public LazyList< Role > Roles
```

Level3: List of roles. Please notice that every user in Magix might belong to more than one role, though there's not UI currently for adding a user into more than one role, meaning defacto there's only one role per user today

[ActiveField()]
private internal LazyList< UserSettings > Settings
Level3: Settings for user instance

public string RolesString
Level3: Will return a string containing all roles user belongs to

public UserBase Current
Level3: Will return or change the currently logged in user object. Notice that Users are in Magix stored in cookies on the harddisks of the browsers that clients visits your website with. These cookies are currently being stored as non-persistive, and HttpOnly, but will make sure that as long as user doesn't close his browser, he will become automatically logged in again upon visiting the site, if he had visited before without closing his browser

124. Role

[ActiveType(TableName="docWineTasting.CoreTypes.Role")]

Magix.Brix.Components.ActiveTypes.Users.Role

Description:

Level3: Contains the roles i the system. Every user belongs to a 'role' which gives him rights in regards to some aspect of functionality or something. All authorization, by default in Magix, will go through this Role class, and which specific roles the currently logged in user belongs to

Basetypes

Magix.Brix.Data.ActiveType-g

Methods

public override void Save()

Level3: Overridden to make sure Name is unique among other things

Properties

[ActiveField()]

public string Name

Level3: The name of the role, typically 'Administrator' or 'Guest' or something

125. Criteria

Magix.Brix.Data.Criteria

Description:

Level3: Abstract base class for all data storage retrieval criterias. Also contains several handy static constructors for easy creation of data storage retrieval criterias.

Methods

public Criteria Eq(string propertyName, object value)

Level3: Static constructor to create a criteria of type Equals.

public Criteria Ne(string propertyName, object value)

Level3: Static constructor to create a criteria of type NotEquals.

public Criteria Like(string propertyName, string value)

Level3: Static constructor to create a criteria of type LikeEquals. Notice that this is equal to the LIKE keyword from SQL, meaning you can use % and _ as control characters to look for 'wild card combinations'. Laymen terms; % == * and _ == ?

public Criteria Id(int id)

Level3: Static constructor to create a criteria of type CritID. Will create a Criteria that will demand the ID of the object is id

public Criteria NotId(int id)

Level3: Static constructor to create a criteria of type CritNoID. Will create a Criteria that will demand the ID of the object is NOT id

```
public Criteria NotLike(string propertyName, string value)
```

Level3: Static constructor to create a criteria of type LikeNotEquals. The opposite of Like

```
public Criteria Lt(string propertyName, object value)
```

Level3: Static constructor to create a criteria of type LessThen. Depends upon the column type, but normally a dash of reason can deduct this, regardless of the column type. String are compared alphabetically, all other from a 'least is less' standpoint. Will return only objects that are 'Less than' the given value

```
public Criteria Mt(string propertyName, object value)
```

Level3: Static constructor to create a criteria of type MoreThen. 'Opposite' of Lt. See Lt

```
public Criteria ParentId(int id)
```

Level3: Static constructor to create a criteria of type ParentIdEquals. Will enforce that the only objects being chosen are all belonging to a parent object, with the given id

```
public Criteria ExistsIn(int id)
```

Level3: Static constructor to create a criteria of type ExistsInEquals. Which will make sure there exists a relationship between the resulting object and the object with the given id in such a way that the other object are 'referencing' this one in one way or another. If you need to express the 'opposite', as in 'return objects who are referencing id object', then use the overloaded version of this method and pass in true as the reversed parameter. This will ensure that only objects which themselves are referencing the id object will be returned. For instance if a Customer has a LazyList of Contacts, you could find all Contacts referenced in a specific customer through using `Contact.Select(Criteria.ExistsIn(customerId))` If you want to go the other way, as in finding all Customers that are referencing a specific contact you'd have to go `Customer.Select(Criteria.ExistsIn(contactId, true))` The above works, as long as the 'IsOwner' parts of a relationship equals false. If it's an 'IsOwner type of relationship', you'd rather using `IsChild` and `IsParent` methods

```
public Criteria ExistsIn(int id, bool reversed)
```

Level3: Static constructor to create a REVERSED criteria of type ExistsInEquals. See the documentation to the overload for an explanation

```
public Criteria HasChild(int id)
```

Level3: Static constructor to create a criteria of type HasChildId. Will only return objects that have the given id as their 'child objects' [meaning 'IsOwner' == true in its ActiveField declaration]

```
public Criteria Sort(string colName)
```

Level3: Static constructor to create a criteria of type Sort. Will sort on the Column name

```
public Criteria Sort(string colName, bool ascending)
```

Level3: Static constructor to create a criteria of type Sort. Will sort on the column name. ascending decides whether or not ascending or descending

```
public Criteria Range(int start, int end, string sortColumn, bool ascending)
```

Level3: Static constructor to create a criteria of type Range. Will first of all sort according to the sortColumn, either ascending or descending depending upon the 'ascending' parameter. Then it will return only the [start, end} result set from that dataset. Executes actually __BLISTERING__ fast, even though you probably wouldn't believe so ... ;) Basically the 'foundation' for the Grid System in many ways ...

Properties

```
public string PropertyName
```

Level3: Name of property associated with criteria.

```
public object Value
```

Level3: Value that criteria associates with the given property of your criteria.

126. Transaction

Magix.Brix.Data.Transaction

Description:

Level3: Implements transactional support for your updates and inserts. Use through the C# using keyword to get automatic rollbacks. Or implement finally yourself in your code. Remember to call 'Commit' before Transaction is lost, since otherwise. Caputt. Default is Rollback

Constructors

```
public Transaction(Adapter ad)
```

Level3: NOT meant for accessing directly

Methods

```
public void Commit()
```

Will Commit the entire batch of jobs, and release the transaction

```
protected void Rollback()
```

Level3: Will do a Rollback on your entire changes to the database. Will also reset the cache

Properties

public abstract IDbTransactionTrans
Level3: NOT meant for accessing directly

127. ActiveTypeAttribute

Magix.Brix.Data.ActiveTypeAttribute

Description:

Level3: Mark your well known types or entity types [or more correctly said; Active Types ;)] with this attribute to make them serializable. In addition you must inherit from ActiveType with the type of the type you're creating as the generic type argument. Notice that this attribute is for classes, you still need to mark every property that you wish to serialize with the ActiveFieldAttribute.

128. ActiveType-g

Magix.Brix.Data.ActiveType-g

Description:

Level3: Inherit your well known types or entity types - the types you want to serialize to your database from this class giving the generic argument type as the type you're creating. Notice that you also need to mark your types with the ActiveRecordAttribute attribute in addition to marking all your serializable properties with the ActiveFieldAttribute.

Basetypes

Magix.Brix.Data.TransactionalObject

Methods

public int CountWhere(params Criteria[])

Level3: Returns the number of objects in your data storage which is of type T where the given criterias are true

public T SelectByID(int id)

Level3: Returns the object with the given ID from your data storage. Will return null if either it's a 'type-clash' or object doesn't exist, which are for all practical concerns for you, and should be, irrelevant

public IEnumerable(T) SelectByIds(params int[])

Level3: Returns a list of objects with the given ID from your data storage

public T SelectFirst(params Criteria[])

Level3: Returns the first object from your data storage which are true for the given criterias. Pass nothing () if no criterias are needed.

```
public IEnumerable( T ) Select(params Criteria[])
```

Level3: Returns all objects from your data storage that matches the given criterias. Pass nothing () if you want all objects regardless of any values or criterias.

```
public bool operator!=(ActiveType( T, ActiveType( T)
```

Level3: Returns true if the two given objects does not have the same ID.

```
public bool operator==(ActiveType( T, ActiveType( T)
```

Level3: Returns true if the two given objects do have the same ID.

```
public void Delete()
```

Level3: Deletes the object from your data storage.

```
public override void Save()
```

Level3: Save the object to your data storage. This should normally be overridden to make sure the end user isn't messing up your domain model somehow

```
public override bool Equals(object obj)
```

Level3: Returns true if the given object have the same ID as the this object.

```
public override string ToString()
```

Level3: Will return the ID of the ActiveType. Little bit of 'debugging helping'

Properties

`public override int ID`

Level3: The data storage associated ID of the object. Often the primary key if you're using a database as your data storage.

`public int Count`

Level3: Returns the number of objects in your data storage which is of type T.

129. ActiveFieldAttribute

Magix.Brix.Data.ActiveFieldAttribute

Description:

Level3: Used to mark entity objects as serializable. If a property is marked with this attribute then it will be possible to serialise that property. Notice that you still need to mark you classes with the ActiveRecordAttribute. Also only properties, and not fields and such can be marked as serializable with this attribute.

130. Adapter

Magix.Brix.Data.Adapter

Description:

Level4: Abstract base class for all Database Adapters in Magix-Brix. If you wish to build your own data adapter then inherit from this class and implement the abstract methods, add up a reference to the dll and change the data-configuration line in your configuration file and it should work. Class provides some common functions needed for all database adapters like caching, instantiation and such in addition.

Methods

public void InvalidateCache()

Level4: Careful here. This is often a VERY expensive operation if you're doing it frequently. Though sometimes needed I guess. This one is mostly here for being able to go 'completely reset' in case of transactional rollbacks and such. If you use this one directly yourself, you'd better be sure you know what you're doing. Since alternatively you'd win the 'slowest system on the planet' award ... ;)

public abstract void ResetTransaction()

Level4: DO NOT TOUCH this one either. It's exclusively here [really] for internal usage in regards to transaction support for rollbacks and such

public object SelectByID(Type type, int id)

Level4: Retrieves an object with the given ID from your data storage.

public void Delete(int id)

Level4: Deletes the object with the given ID from your data storage.

public abstract int CountWhere(Type type, params Criteria[])

Level4: Should return the number of items of the given type from your data storage with the given criterias.

public abstract object SelectFirst(Type type, string propertyName, params Criteria[])

Level4: Should return the first object of type; "type" - with the given criterias.

public abstract IEnumerable(object) Select(Type type, string propertyName, params Criteria[])

Level4: Should return all objects of the given type with the given criterias.

public abstract string GetConnectionString()

Level4: Should return some sort of string identification of the underlying datasource

public abstract Transaction BeginTransaction()

Level4: Begins a new transaction object, which ensures the entire operation within the scope of the transaction object will be either saved or rejected, and thrown an exception from ...

public abstract IEnumerable(object) Select()

Level4: Should return all objects in your data storage.

public abstract void Save(object value)

Level4: Should save the given object into your data storage.

public abstract void Close()

Level4: Called when data storage should close. Often used to close file handles or database connections etc.

public abstract void Open(string connectionString)

Level4: Called when your data storage should be opened. Often used to open file handles or database connections etc.

protected abstract object SelectObjectByID(Type type, int id)

Level4: Should return the object from your data storage with the given ID being of the given Type.

protected abstract void DeleteObject(int id)

Level4: Should delete the object with the given ID from your data storage.

Properties

public AdapterInstance

Level4: Retrieves the configured database adapter. Notice that you would very rarely want to use this directly but instead access it indirectly through your ActiveType classes

public List< Type > ActiveTypes

Level4: A list of all your ActiveTypes in the system

public List< Type > ActiveModules

Level4: A list of all your ActiveModules in the system

public Dictionary< int, object > Cache

Level4: Dictionary of ID, ActiveType objects built up during the Request. Every ActiveType object being fetched during on HTTP request [one postback] is being held in a cache for the duration of the rest of that request on a 'per request basis'. This is because research have shown that even though completely different modules, unknown to each other really, are constantly running in paralel, they often tend to react upon the same actual list of objects. Meaning that by caching everything on one request like this we loose basically nothing, since we're in GC land anyway, and aren't really 'locking up memory' in anyways. While we also get to have a __BLISTERING__ fast DataAdapter technology, since after it's in the cache, the criterias will only execute to fetch the ID of the object you're requesting, if you're using 'complex queries', and if you're querying directly for the ID's, you'll get immediately here upon requesting your ActiveTypes. Hence; Hoahh ...! SWOSH ...! :P

131. TransactionalObject

Magix.Brix.Data.TransactionalObject

Description:

Baseclass for 'internal usage' behind ActiveTypes. Definitely candidate for refactoring. In general do NOT reference any members from here directly, except the ID of course

132. MSTransaction

Magix.Brix.Data.Adapters.MSSQL.MSTransaction

Description:

Level4: Implementation of Transaction class for MS SQL DataAdapter

Basetypes

Magix.Brix.Data.Transaction

Methods

public override void Commit()

Will Commit the entire batch of jobs, and release the transaction

protected override void Rollback()

Level3: Will do a Rollback on your entire changes to the database. Will also reset the cache

Properties

public override IDbTransaction Trans

Level3: NOT meant for accessing directly

133. MSSQL

Magix.Brix.Data.Adapters.MSSQL.MSSQL

Description:

Level4: Microsoft SQL Server Database Adapter, which probably works with 2005 and any later, for Magix-Brix. Contains all MS SQL specific logic needed to use Magix-Brix together with MS SQL. This is the default DataAdapter in use in Magix

Basetypes

Magix.Brix.Data.Internal.StdSQLDataAdapter

Magix.Brix.Data.IPersistViewState

Methods

public override void Open(string connectionString)

Level4: Opens a connection to the MS SQL database

public override Transaction BeginTransaction()

Level4: Will create a new Transaction and return to caller

public override int CountWhere(Type type, params Criteria[])

Level4: Should return the number of items of the given type from your data storage with the given criterias.

public override object SelectFirst(Type type, string propertyName, params Criteria[])

Level4: Should return the first object of type; "type" - with the given criterias.

```
public override IEnumerable( object ) Select(Type type, string propertyName, params Criteria[])
```

Level4: Should return all objects of the given type with the given criterias.

```
public override IEnumerable( object ) Select()
```

Level4: Should return all objects in your data storage.

```
public override void Save(object value)
```

Level4: Should save the given object into your data storage.

```
public override void Close()
```

Level4: Called when data storage should close. Often used to close file handles or database connections etc.

```
public void Save(string sessionId, string pageUrl, string content)
```

Level4: Called by Magix when it's time to save the ViewState. Normally expected to dump the stuff into a DB somewhere with the key of sessionId+pageUrl or something

```
public string Load(string sessionId, string pageUrl)
```

Level4: Called by Magix when it's time to reload the ViewState. Normally expected to dump the stuff out of a DB somewhere with the key of sessionId+pageUrl or something

```
public override void ResetTransaction()
```

Level4: DO NOT TOUCH this one either. It's exclusively here [really] for internal usage in regards to transaction support for rollbacks and such

```
public override string GetConnectionString()
```

Level4: Should return some sort of string identification of the underlying datasource

protected override object SelectObjectByID(Type type, int id)

Level4: Should return the object from your data storage with the given ID being of the given Type.

protected override void DeleteObject(int id)

Level4: Should delete the object with the given ID from your data storage.

134. Helpers

Magix.Brix.Data.Internal.Helpers

Description:

Level3: Static helper class for data-storage Adapter developers. If yo're fiddling around here, you'd better know what you're doing ...!! ;)

135. StdSQLDataAdapter

Magix.Brix.Data.Internal.StdSQLDataAdapter

Description:

Level4: Common logic for all Database adapters that relies on standard SQL syntax. Wrappers for creating SQL text for derived data adapters that uses RDBS that relies on standard SQL syntax. Sorry, I have to prioritize what I comment. Needs refactoring though, that's one thing but so does the entire Magix.Data namespace ... :(

Basetypes

Magix.Brix.Data.Adapter

136. ActiveModule

Magix.Brix.Loader.ActiveModule

Description:

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your Modules, such as RaisingEvents etc. Inherit your ActiveModules from this class to simplify their usage

Basetypes

Magix.Brix.Loader.IModule

Methods

protected delegate void executor()

Level3: Helper for executing 'dangerous code' such that if an exception happens, it'll 'swallow' the exception, and show a Message box showing the exception

protected Node RaiseEvent(string eventName)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s)

protected void RaiseEvent(string eventName, Node node)

Level3: Shorthand for raising events

protected bool RaiseSafeEvent(string eventName, Node node)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s). This method will trap any exceptions occurring, and show a message box back to the end user with its exception content, if any

protected void ShowMessage(string body)

Level3: Shows a Message to the user with the given body

protected void ShowMessage(string body, string header)

Level3: Shows a Message to the user with the given body and header

protected void ShowMessage(string body, string header, int milliseconds)

Level3: Shows a Message to the user with the given body and header and Milliseconds

protected void ShowError(string body, string header)

Level3: Shows an Error Message to the user with the given body and header

protected void IncludeCssFile(string file)

Level3: Will include the given CSS file onto the page. Useful for injecting your own CSS files onto the page

protected string GetApplicationBaseUrl()

Level3: Will return the 'base' URL of your application. Meaning if your application is installed on x.com/f then x.com/f will always be returned from this method. Useful for using as foundation for finding specific files and so on

protected bool ExecuteSafely(executor functor, string msg, params object[])

Level3: Helper for execute code that you suspect might throw exceptions. Will trap exception and show a message box back to end user instead of allowing exception to penetrate through to Yellow Screen of Death. Will return true if operation didn't throw an exception and false if it did throw an exception

public void InitialLoading(Node node)

Level3: You Must call this one, if you override it, to set the Module's DataSource property. And in order to get some other types of wiring running correctly for the module

```
public Node RaiseSafeEvent(string eventName)
```

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s). This method will trap any exceptions occurring, and show a message box back to the end user with its exception content, if any

Properties

```
protected      bool    FirstLoad
```

Level3: Is true if this was the request when the Module was loaded initially, somehow

```
protected      Node    DataSource
```

Level3: The Node passed into InitialLoading will automatically be stored here ...

137. ActiveEvents

Magix.Brix.Loader.ActiveEvents

Description:

Level3: Class contains methods for raising events and other helpers, like for instance helpers to load controls and such. Though often you'll not use this directly, but rather use it through helper methods on your ActiveControllers and ActiveModules

Methods

```
public void RaiseLoadControl(string name, string position)
```

Level3: Loads a control with the given name (class name) into the given position (name of Magix.UX.Dynamic in the Viewport currently used). Use this method to load Modules. Notice that there exists an overload of this method which takes an object parameter that will be passed into the InitialLoading method when control is loaded.

```
public void RaiseLoadControl(string name, string position, Node parameters)
```

Level3: Loads a control with the given name (class name) into the given position (name of Magix.UX.Dynamic in the Viewport currently used). Use this method to load Modules. This overload of the method will pass the "initializingArgument" parameter into the InitialLoading method when control is loaded.

```
public void RaiseClearControls(string position)
```

Level3: Clear all controls out of the position (Magix-Dynamic) of your Viewport.

```
public void RaiseActiveEvent(object sender, string name)
```

Level3: Raises an event with null as the initialization parameter. This will dispatch control to all the ActiveEvent that are marked with the Name attribute matching the name parameter of this method call.

```
public void RaiseActiveEvent(object sender, string name, Node pars)
```

Level3: Raises an event. This will dispatch control to all the ActiveEvent that are marked with the Name attribute matching the name parameter of this method call.

```
public void CreateEventMapping(string from, string to)
```

Level3: Will override the given 'from' ActiveEvent name and make it so that every time anyone tries to raise the 'from' event, then the 'to' event will be raised instead. Useful for 'overriding functionality' in Magix. This can also be accomplished through doing the mapping in the system's web.config file

```
public void CreateModuleMapping(string from, string to)
```

Level3: Will create a mapping from the 'from' module to the 'to' module

```
public void RemoveMapping(string key)
```

Level3: Will destroy the given [key] Active Event Mapping

```
public void RemoveModuleMapping(string key)
```

Level3: Will destroy the given [key] Active Module Mapping

```
public string GetEventMappingValue(string key)
```

Level2: Will return the given Value for the given Key Active Event Override

```
public string GetModuleMappingValue(string key)
```

Level2: Will return the given Value for the given Key Active Module Override

Properties

`public ActiveEvents Instance`

Level3: This is our Singleton to access our only ActiveEvents object. This is the property you'd use to gain access to the only existing ActiveEvents object in your application pool

`public IEnumerable< string > EventMappingKeys`

Level3: Returns an Enumerable of all the Keys in the Event Mapping Collection. Basically which Active Events are overridden

`public IEnumerable< string > EventMappingValues`

Level3: Returns an Enumerable of all the Values in the Event Mapping Collection. Basically where the Overridden Active Events are 'pointing'

138. ActiveModuleAttribute

Magix.Brix.Loader.ActiveModuleAttribute

Description:

Level3: Mark your Active Modules with this attribute. If you mark your Modules with this attribute you can load them using the PluginLoader.LoadControl method. This is the main attribute for being able to create ActiveModules

139. ActiveControllerAttribute

Magix.Brix.Loader.ActiveControllerAttribute

Description:

Level3: Mark your controllers with this Attribute. Notice that an Active Controller must have a default constructor taking zero parameters. This constructor should also ideally execute FAST since all controllers in your Magix-Brix project will be instantiated once every request.

140. ActionController

Magix.Brix.Loader.ActiveController

Description:

Level3: Helper class for simplifying some of the common tasks you'd normally want to use from your controllers, such as Loading Modules, raising events etc. Inherit your controllers from this class if you'd like to add more 'power' to them

Methods

protected delegate void executor()

Level3: Helper for executing 'dangerous code' such that if an exception happens, it'll 'swallow' the exception, and show a Message box showing the exception

protected Node LoadModule(string name)

Level3: Loads the given module and puts it into your default container

protected bool ExecuteSafely(executor functor, string msg, params object[])

Level3: Helper for execute code that you suspect might throw exceptions. Will trap exception and show a message box back to end user instead of allowing exception to penetrate through to Yellow Screen of Death. Will return true if operation didn't throw an exception and false if it did throw an exception

protected Node LoadModule(string name, string container)

Level3: Loads the given module and puts it into the given container. Will return the node created and passed into creation

protected void LoadModule(string name, string container, Node node)

Level3: Shorthand method for Loading a specific Module and putting it into the given container, with the given Node structure.

protected Node RaiseEvent(string eventName)

Level3: Shorthand for raising events. Will return a node, initially created empty, but passed onto the Event Handler(s)

protected void RaiseEvent(string eventName, Node node)

Level3: Shorthand for raising events.

protected void ShowMessage(string body)

Level3: Shows a Message to the user with the given body

protected void ShowMessage(string body, string header)

Level3: Shows a Message to the user with the given body and header

protected void ShowMessage(string body, string header, int milliseconds)

Level3: Shows a Message to the user with the given body and header and Milliseconds

protected void ShowError(string body, string header)

Level3: Shows an Error Message to the user with the given body and header

protected void IncludeCssFile(string file)

Level3: Will include the given CSS file onto the page

protected string GetApplicationBaseUrl()

Level3: Will return the 'base' URL of your application

Properties

protected Page Page

Level3: Shorthand for getting access to our "Page" object.

141. ActiveEventAttribute

Magix.Brix.Loader.ActiveEventAttribute

Description:

Level3: Mark your methods with this attribute to make then handle Magix.Brix Active Events. The Name property is the second argument to the RaiseEvent, or the "name" of the event being raised. You can mark your methods with multiple instances of this attribute to catch multiple events in the same event handler. However, as a general rule of thumb it's often better to have one method handling one event

142. ActiveEventArgs

Magix.Brix.Loader.ActiveEventArgs

Description:

Level3: EventArgs class that will be passed into your Magix-Brix events - the methods you mark with the ActiveEvent Attribute. The Extra property will contain the "initializationObject" passed into the RaiseEvent.

Properties

public string Name

Level3: The name of the Active Event. Most Active Event Handlers will be mapped only to one Active Event, but occasionally you'll have one Event Handler handling more than one Event. For cases like this the Name property might be useful to understand which event you're actually handling

public Node Params

Level3: This is the "initializationObject" passed into your RaiseEvent call. Use this parameter to pass around data between components

143. PluginLoader

Magix.Brix.Loader.PluginLoader

Description:

Level4: Helps load UserControls embedded in resources. Relies on that Magix.Brix.Loader.AssemblyResourceProvider is registered as a Virtual Path Provider in e.g. your Global.asax file. Use the Instance method to access the singleton object, then use the LoadControl to load UserControls embedded as resources. Kind of like the Magix' version of Page.LoadControl. Can be used directly by you, if you really know what you're doing though. In general, I'd say DON'T ...!!

Methods

```
public Control LoadActiveModule(string fullTypeName)
```

Level3: Dynamically load a Control with the given FullName (namespace + type name). This is the method which is internally used in Magix-Brix to load UserControls from embedded resources and also other controls. Since ActiveEvents might be mapped and overridden, you actually have no guarantee of that the event you wish to raise is the one who will become raised

Properties

```
public List< Type > ActiveControllers
```

Level4: A list of all your ActiveControllers in the system

```
public PluginLoader Instance
```

Level4: Singleton accessor. Allows access to the 'one and only' PluginLoader


```
public List< Assembly > PluginAssemblies
```

Level4: Will return all assemblies within your Application Pool, minus 'system assemblies'. Useful for 'meta stuff'

144. AssemblyResourceProvider

Magix.Brix.Loader.AssemblyResourceProvider

Description:

Level4: Helper class to make it possible to load controls (and more importantly) UserControls which are embedded as resources in DLLs. Not intended for direct usage, but will be in 'the background' and making sure you can load ActiveModules as resources from your DLLs

145. PublisherPluginAttribute

Magix.Brix.Publishing.Common.PublisherPluginAttribute

Description:

Level3: I'd be highly surprised if this is not your first entry to Magix in C#. This is the PublisherPlugin attribute, which you can use to create your own plugins for the Publishing system within. Implement this attribute on your ActiveModules and VOILA! They'll surface up as selections in your WebPageTemplate editing operations and be usable as plugins in your system. Probably the easiest way on the planet to create a plugin for any kind of system out there. Especially in combination with the logic behind the ModuleSettingAttribute

146. ModuleSettingAttribute

Magix.Brix.Publishing.Common.ModuleSettingAttribute

Description:

Level3: Wraps a setting property for a PublisherPlugin

147. PeriodCollection

Magix.Brix.Types.PeriodCollection

Description:

Level3: A collection class of Period types. Contains algebraic methods for OR, AND, XOR, NOT. Makes algebraic operations on collection of Period objects very easy and intuitive. Can with for instance one line of code OR two collections together to find the logically OR'ed result of these two different collections. Very useful for manipulating dates and such

Constructors

```
public          PeriodCollection()  
Level3: Default CTOR
```

```
public          PeriodCollection(IEnumerable< Period>  
Level3: Initializes the list with the given collection
```

Methods

```
public void Normalize()
```

Level3: Sorts and runs through all period objects and checks for overlapping, eliminating overlapping periods merging them into one and such. Kind of like the same mathematical operation as normalizing a vector. Crucial for most of the algebraic operations. This operation will most often make the containing number of items less then it used to be before the operation. Meaning two Periods that overlap will become one 'combined' period after this method is done

`public PeriodCollection NOT()`

Level3: Returning the NOT operation of the this list. Basically all the places where there are NO Periods within the collection of Periods. Kind of the "negative" of the collection. Notice that since we do NOT treat a PeriodCollection as an "open end collection" but rather as a min value of DateTime.MinValue and a max value of DateTime.MaxValue we do not get the "elephant footprint" as we would otherwise be forced to have. Meaning that the operation is 'reversible' by calling it twice.

`public TimeSpan Sum()`

Level3: Returns the total amount of time in the this collection

`public void Trim(Period period)`

Level3: Trimming away everything that's not within given Period

`public void AddRange(IEnumerable< Period>)`

Level3: Adds the given range of periods into the collection

`public void Sort(Comparison< Period>)`

Level3: Takes a predicate and sort the list accordingly

`public void Sort()`

Level3: Sorts according to a predicate that sorts first prioritized after start of periods and then according to end of periods if start of periods are the same. Used in the Normalize method

`public Period Find(Predicate< Period>)`

Level3: Takes a predicate and returns the first Period that matches the predicate

`public List< Period > FindAll(Predicate< Period>)`

Level3: Returns a list of Periods that matches the predicate

```
public void RemoveAll(Predicate< Period>
```

Level3: Removes all the periods that matches the predicate

```
public int IndexOf(Period item)
```

Level3: Returns the index of the given item, or -1 if no found

```
public void Insert(int index, Period item)
```

Level3: Inserts the given period at the given index

```
public void RemoveAt(int index)
```

Level3: Removes the period at the given index

```
public void Add(Period item)
```

Level3: Appends to the back of the list the given period.

```
public void Clear()
```

Level3: Completely clears all periods from the list

```
public bool Contains(Period item)
```

Level3: Returns true if the given period is found in the list

```
public void CopyTo(Period[] array, int arrayIndex)
```

Level4: Copy the list of periods into the given array starting at arrayIndex in the collection

```
public bool Remove(Period item)
```

Level3: Removes the given period from the collection

```
public IEnumerator( Period ) GetEnumerator()
```

Level3: Enumerating support

```
public PeriodCollection OR(PeriodCollection left, PeriodCollection right)
```

Level3: Returns the logically ORed lists back to caller. Does not in any ways change the given lists.

```
public PeriodCollection AND(PeriodCollection left, PeriodCollection right)
```

Level3: Logically ANDs two collections together returning the ANDed result. Basically a new collection of Period objects which consists of the date ranges that overlaps within both lists. Useful for finding out for instance when two period collections overlap

```
public PeriodCollection XOR(PeriodCollection left, PeriodCollection right)
```

Level3: Returning the logical XOR of two given collections. The XOR result is the place where ONE and ONE ONLY of the given collections have values. This will normally increase the number of periods in your collection. Doesn't change the incoming periods [left, right]

Properties

```
public DateTime Starts
```

Level3: Returns the lowest start date of the collection

```
public DateTime Ends
```

Level3: Returns the highest end date of the collection

`public Period this[int index]`

Level3: Returns the period at the given index. Will throw if out of bounds. Setter will replace the existing period at the given index.

`public int Count`

Level3: Returns the number of items in the collection

148. LazyList-g

Magix.Brix.Types.LazyList-g

Description:

Level3: Helper class for Lazy Loading of Child ActiveTypes objects. Basically just a list generic type, that'll not load objects before needed. Useful for using as properties for list of child objects in your ActiveTypes

Methods

public bool Exists(Predicate(T)

Level3: Traverses the list and returns true if the item you're looking for exists, as in the predicate returns true

public T Find(Predicate(T)

Level3: Traverses the list and returns the first item matching the predicate

public IEnumerable(T) FindAll(Predicate(T)

Level3: Traverses the list and returns all items matching the predicate

public void AddRange(IEnumerable(T)

Level3: Adds a range of new items to the list collection

public void RemoveAll(Predicate(T)

Level3: Removes all items matching the given predicate

public int IndexOf(T item)

Level3: Returns the index of the given item, if it exists in the list

```
public void Insert(int index, T item)
```

Level3: Inserts a new item at the given position

```
public void RemoveAt(int index)
```

Level3: Removes the item at the given index

```
public void Add(T item)
```

Level3: Appends a new item to the list

```
public void Clear()
```

Level3: Clears the list of all its items

```
public bool Contains(T item)
```

Level3: Returns true if the specific item exists in the list

```
public void CopyTo(T[] array, int arrayIndex)
```

Level4: Copies the list to the given array, starting at the given offset

```
public bool Remove(T item)
```

Level3: Removes the specific item from the list, returns true if an item was removed. Returns false if the item doesn't exist in the list

Properties

`public bool ListRetrieved`

Level3: Returns true if list is already populated. If the LazyList is not retrieved, then the child objects in that property won't be saved when saving the parent object. Which can be very useful for optimizing your application. Be careful with 'de-referencing' LazyList properties because of this, unless you really have to

`public T this[int index]`

Level3: Gets or sets the item at the specific index

`public int Count`

Level3: Returns the number of items in our list

`public bool IsReadOnly`

Will return false, always!

149. Node

Magix.Brix.Types.Node

Description:

Level3: Helper class to pass around data in a "JSON kind of way" without having to convert to JSON strings. Create a new instance, and just start appending items to it like this; `Node n = new Node(); n["Customer"]["Name"] = "John Doe"; n["Customer"]["Adr"] = "NY";` This is at the core of Magix, being the 'protocol' we're using to pass data around within the system. If you don't understand this class, you're in trouble! Make sure you understand, at least roughly, what this class does if you'd like to code C# for Magix

Constructors

`public Node()`

Default CTOR, creates a new node with no name and no value and no children

`public Node(string name)`

Creates a new node with the given name

`public Node(string name, object value)`

Creates a new node with the given name and the given value

Methods

`public void SetParent(Node n)`

Level3: Assigns a Parent to the current node. Useful for moving nodes around

```
public Node Find(Predicate( Node))
```

Level3: Returns the first node that matches the given Predicate. Will search recursively. Be careful here, if you're dereferencing nodes that don't exist inside your function, you might very well never return from this method ... ;)

```
public bool Exists(Predicate( Node))
```

Level3: Returns true if node exists as a direct child only, and not search recursive

```
public bool ExistsDeep(Predicate( Node))
```

Level3: Returns true if node exists anywhere as a child or child of child

```
public Node UnTie()
```

Level3: Will "disconnect" the node from its parent node. Useful for removing nodes and trees out of Node structures

```
public int IndexOf(Node item)
```

Level3: Returns the index of the given item, if it exists within the children collection. Otherwise it returns -1

```
public void Insert(int index, Node item)
```

Level3: Inserts a new item into the children collection

```
public void RemoveAt(int index)
```

Level3: Removes node at given index

```
public void Add(Node item)
```

Level3: Adds a new node to the collection

```
public void AddForceNew(Node item)
```

Level3: Adds a new node to the collection

```
public void AddRange(IEnumerable( Node))
```

Level3: Adds a range of nodes to collection

```
public void Clear()
```

Level3: Entirely empties the collection

```
public bool Contains(Node item)
```

Level3: Returns true if node exists within child collection [flat]

```
public bool Contains(string itemName)
```

Level3: Returns true if node exists within child collection [flat]

```
public bool Remove(Node item)
```

Level3: Removes the given node from the child collection

```
public IEnumerator( Node ) GetEnumerator()
```

Level3: Supports enumerating items

```
public override string ToString()
```

Level4: Will return name/value and number of children as a string

```
public string ToJSONString()
```

Level3: Will translate the Node structure to a JSON string. Useful for passing stuff around to other systems, and integrating with client-side etc. Be warned! No TYPE information is being passed, so you cannot build the same Node structure by reversing the method and call FromJSON after creating JSON out of your node

```
public void Sort(Comparison< Node>)
```

Level3: Will sort the nodes according to your given comparison delegate

```
public Node RootNode()
```

Level3 Returns the outer most parent node, the top node of the hierarchy

```
public Node Clone()
```

Level3: Clones the given node. Deep copy of all nodes

```
public Node FromJSONString(string json)
```

Level3: Will de-serialize the given JSON string into a Node structure. PS! Even though Nodes can be serialized to JSON, the type information is lost, meaning you can not go back again 100% correctly, since you're 'loosing your types' when converting from Node to JSON. This is on our road map for fixing, but currently not finished

Properties

```
public Node Parent
```

Level3: Returns the Parent node of the current node in the hierarchy. Normally you'd never need to know the 'Parent' of a node, due to the intrinsic logic of Magix, so be careful. If you're using this method, you're probably doing something wrong on an architectural level. Be warned ...

public string Name

Level3: Returns the name of the node

public object Value

Level3: Returns the value of the object

public Node this[string name]

Level3: Returns the node with the given Name. If that node doesn't exist a new node will be created with the given name and appended into the Children collection and then be returned. Please notice that this method CREATES NODES during DE-REFERENCING. And it is its INTENT TOO! ;)

public Node this[int index]

Level3: Returns the n'th node

public int Count

Level3: Returns the number of items in the children collection

150. SingleContainer

[ActiveModule]

Magix.Brix.Viewports.SingleContainer

Description:

Level2: Contains the logic for the main Viewport in Magix. A viewport can be seen as your 'design' and contains all the different logic for being able to load and unload modules and such

Basetypes

Magix.Brix.Loader.ActiveModule

Methods

private new void IncludeCssFile(string cssFile)

Level3: Will include the given CSS file onto the page. Useful for injecting your own CSS files onto the page

[ActiveEvent(Name = "")]

protected void NULLEventHandler(object sender, ActiveEventArgs e)

Level2: Handled to make sure we log our events into the Debug window if enabled

[ActiveEvent(Name = "Magix.Core.SetTitleOfPage")]

protected void Magix_Core_SetTitleOfPage(object sender, ActiveEventArgs e)

Level2: Will set the Title element of the page to the given 'Caption'

[ActiveEvent(Name = "Magix.Core.SetViewPortContainerSettings")]

protected void Magix_Core_SetViewPortContainerSettings(object sender, ActiveEventArgs e)

Level2: Will change the Viewport's settings such as CSS class, margins, size etc. Legal parameters are 'Width', 'Top', 'MarginBottom', 'PullTop', 'Height', 'PushLeft', 'PushRight', 'Padding', 'Last' and 'CssClass'

```
[ActiveEvent(Name = "Magix.Core.GetViewPortSettings")]  
protected void Magix_Core_GetViewPortSettings(object sender, EventArgs e)
```

Level2: Will return the settings for the Viewport back to caller. 'Width', 'Top', 'MarginBotto', 'Last' and so on

```
[ActiveEvent(Name = "Magix.Core.AddLinkInHeader")]  
protected void Magix_Core_AddLinkInHeader(object sender, EventArgs e)
```

Level2: Will add a 'link header element' to your rendered HTML

```
[ActiveEvent(Name = "Magix.Core.PlaySound")]  
protected void Magix_Core_PlaySound(object sender, EventArgs e)
```

Level2: Will play the given 'File' as a sound. Supported sound formats depends upon browsers, but normally .ogg is pretty safe

```
[ActiveEvent(Name = "Magix.Core.PauseSound")]  
protected void Magix_Core_PauseSound(object sender, EventArgs e)
```

Level2: Will stop, or pause, any sound or music immediately

```
[ActiveEvent(Name = "Magix.Core.AddCustomCssFile")]  
protected void Magix_Core_AddCustomCssFile(object sender, EventArgs e)
```

Level2: Injects a CSS file onto the page for inclusion on the client side for you

```
[ActiveEvent(Name = "Magix.Core.AddCustomJavaScriptFile")]  
protected void Magix_Core_AddCustomJavaScriptFile(object sender, EventArgs e)
```

Level2: Injects a CSS file onto the page for inclusion on the client side for you

```
[ActiveEvent(Name = "Magix.Core.ShowMessage")]  
protected void Magix_Core_ShowMessage(object sender, EventArgs e)
```

Level2: Will show a 'Message Box' with your 'Message', 'Header' for 'Milliseconds' time period. If 'IsError' is true, it'll be red and contain some 'error logic' within it. If 'Delayed' is true, the message will not be shown directly, but in fact 'postponed' to the next request. Which can be useful for e.g. Async event handlers, needing to tell the user something, or when you're redirecting the user, but need to explain him why and such

```
[ActiveEvent(Name = "Magix.Core.ClearViewportContainer")]  
protected void Magix_Core_ClearViewportContainer(object sender, EventArgs e)
```

Level2: Will clear the incoming 'Position' container for controls, and unload and clean up everything in regards to any modules within that container

```
[ActiveEvent(Name = "Magix.Core.GetNumberOfChildrenOfContainer")]  
protected void Magix_Core_GetNumberOfChildrenOfContainer(object sender, EventArgs e)
```

Level2: Will return the number of Active Modules [or controls] a specific Viewport Container contains. Useful for determining if a specific container is available or not

```
[ActiveEvent(Name = "Magix.Core.LoadActiveModule")]  
protected void Magix_Core_LoadActiveModule(object sender, EventArgs e)
```

Level2: Handled to make it possible to load Active Modules into this Viewport's containers

```
[ActiveEvent(Name = "Magix.Core.SetAjaxWaitImage")]  
protected void Magix_Core_SetAjaxWaitImage(object sender, EventArgs e)
```

Allows you to change the 'Ajax Wait Image' for your application. Every time an Ajax request [something is clicked e.g.] is sent to the server, a Please Wait "Window" will display an animated image while you're waiting. If you wish to change this animated Image, you can raise the 'Magix.Core.SetAjaxWaitImage' event, which will update the image according to the new 'Image' value passed in

151. AspectModal

Magix.UX.Aspects.AspectModal

Description:

Aspect useful for making 'modal' controls. A Modal control will obscure the contents behind it by creating a div that is semi-transparent (most often) and fill the entire viewport (browser area). This effect makes it impossible to click controls behind it on the page.

Basetypes

Magix.UX.Aspects.AspectBase

Methods

protected override string GetClientSideScript()

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

Properties

public decimal Opacity

The transparency of the control. If this value is 1 the 'modality-div' created will be 100% opaque. If 0.5 it will be 50% transparent, if 0 it will be 100% transparent, but still actually do its job to some extent.

public Color Color

The color of the 'modality-div'. The default value is 'Black'. If you also set its BottomColor property, then a gradient will be created between the Color (top parts) and the BottomColor (bottom) and it will gradient between those starting from the top.

public Color BottomColor

The color of the 'modality-div'. The default value is 'Black'. If you also set its BottomColor property, then a gradient will be created between the Color (top parts) and the BottomColor (bottom) and it will gradient between those starting from the top.

152. AspectAjaxWait

Magix.UX.Aspects.AspectAjaxWait

Description:

Ajax Wait Aspect. Use this one whenever you've got an Ajax Control that you know will spend a lot of time on the server-side executing its logic. You can either have the client-side logic create its own DOM element to serve as a 'blackout' when it's supposed to display the Ajax Wait DOM element, or you can attach an existing in-visible DOM element to it by using the Element property.

Basetypes

Magix.UX.Aspects.AspectBase

Methods

protected override string GetClientSideScript()

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

Properties

public int Delay

The number of milliseconds that will pass after an Ajax Request has been initiated before the Ajax Wait obscurer will be shown. The default value is 500.

`public decimal Opacity`

The opacity or transparency of the obscurer DOM element. The default value is 0.5 which means 50% transparent.

`public string Element`

The Element you've attached to this particular Ajax Wait. If none is given, Magix UX will on the client-side automatically create a black DOM element for you which will serve as the obscurer.

153. AspectBase

Magix.UX.Aspects.AspectBase

Description:

Base class for all Aspects in Magix UX. Implements common functionality.

Basetypes

Magix.UX.Widgets.Core.BaseControl

Methods

protected override void RenderJson(HtmlTextWriter writer)

Only override this one if you truly know what you're doing, and you really need some modified logic. See how the Aspects work for a working example of how this method can be overridden.

public void EnsureViewStateLoads()

Helper methods for Aspect developers. Loads state from the client-side into the Control/Aspect. Used in e.g. AspectDraggable for changing the top/left property of the control when moved and its got a server-side binding (event handler)

154. AspectDraggable

Magix.UX.Aspects.AspectDraggable

Description:

Aspect for creating controls that can be moved around on the screen using your mouse. It is also possible to handle the Dragged event and track when the dropping of the control occurs.

Basetypes

Magix.UX.Aspects.AspectBase

Methods

`public override void EnsureViewStateLoads()`

Helper methods for Aspect developers. Loads state from the client-side into the Control/Aspect. Used in e.g. AspectDraggable for changing the top/left property of the control when moved and its got a server-side binding (event handler)

`public override void RaiseEvent(string name)`

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

`public EventHandler Dragged`

Handle this event to track when the control is dragged and dropped by the user. If this one is omitted, there will not be raised an Ajax Request at all when the control is moved, and hence the ViewState value of the top/left corner of the control will never be updated.

Properties

`public Rectangle Bounds`

The bounding rectangle that the control is possible to move within. Rectangular constraints. The default is 'no constraints'.

`public Point Snap`

The grid that the control will be moved within. If this one is set to 50,50 then the control will snap into a 'grid' of 50 pixels large areas. Useful for making sure a control is being moved only within some rectangular constraints.

`public string Handle`

The handle from which the control itself will be able to drag from. If omitted you can drag the control from any place inside of the control. If a handle is defined, then the control can only be dragged from this specific handle. The Window control is a good example of utilizing this logic. You can drag the entire Window, which means the Window has an AspectDraggable associated with it. But you can only drag the Window by its Caption, which is the handle for the AspectDraggable within the Window.

`public bool Enabled`

If this one is false, then the control cannot be dragged. Useful for having an `AspectDraggable` which might be enabled or not on the fly. The `Window` is a good example of this. It always have the `AspectDraggable` associated with it, but if the `Window` is being set to 'non-movable', then the `AspectDraggable` will be disabled.

155. Uploader

Magix.UX.Widgets.Uploader

Description:

Level4: Widget encapsulating an HTML5 drag and drop File Uploader control, with progree bar and support for multiple files

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

public string GetFileRawBASE64()

Level4: Returns the raw BASE64 encoded information about the file. Use Convert.FromBase64... to convert file to raw byte[] content ...

public string GetFileName()

Level4: Returns the file's client side filename

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

`protected override string GetClientSideScriptType()`

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

`protected override void AddAttributes(Element el)`

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

`protected override void RenderMuxControl(HtmlBuilder builder)`

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

`public EventHandler Uploaded`

Level4: Fired when File is uploaded. Access the file in raw [BASE64 encoded] through the GetFileRawBASE64

Properties

`public int SizeOfBatch`

Level4: Since the user can drag and drop multiple files into the browser at the same time, we need to track which is our 'current file' to be able to notify our code that a 'batch is finished'. This one returns the size [number of files] in the 'current processed batch'. Meaning, as long as `SizeOfBatch` is higher than `CurrentNo`, then you've got more files coming in after this one ...

`public int CurrentNo`

Level4: Since the user can drag and drop multiple files into the browser at the same time, we need to track which is our 'current file' to be able to notify our code that a 'batch is finished'. This one returns the current index in the 'current processed batch'. Meaning, as long as `SizeOfBatch` is higher than `CurrentNo`, then you've got more files coming in after this one ...

156. SlidingMenu

Magix.UX.Widgets.SlidingMenu

Description:

This widget is a 'menu looka-like kinda control'. Its purpose is to be an alternative way of displaying hierarchical choices for your users. But instead of 'popping up' like menu items does in a conventional menu, they will 'slide in' from the side, making a much more efficient way of handling your space on your web pages. For most navigational purposes, this widget is a far better bet than the conventional Menu widget.

Basetypes

Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler MenuItemClicked

Raised when any SlidingMenuItem is clicked within the SlidingMenu hierarchy. Notice that if you actually choose to create an event handler for this event, then anything you do will trigger a server-request, and hence your application will not be as responsive. You might, in most cases, choose to only implement event handlers for the LeafMenuItemClicked instead of handling this event handler.

`public EventHandler LeafMenuItemClicked`

Raised when a 'leaf menu item' is clicked. A Leaf MenuItem is a menu item without any children. Since most menu items which have children are used as groupings, and not really all that interesting to get notifications about when clicked, this is probably the event you wish to handle, unless all your menu items are URL menu items, in which case you don't even need to handle this event.

Properties

`public string ActiveMenuItem`

The active SlidingMenuItem of your SlidingMenu. Use this one to see which menu item was clicked in for instance your event handlers for menu item clicked.

157. SelectList

Magix.UX.Widgets.SelectList

Description:

A DropDown list type of control, although it can also be set into a non-drop down mode. Basically multiple choices type of widget. To some extent, it overlaps the logical functionality of the RadioButton widget, although the SelectList is more useful for cases where you have a massive number of choices, like for instance choose one out of 300 different languages, while the RadioButton is more useful for cases where you have fewer choices, such as choose 'coffee, tea or water'. Add up your choices by adding up ListItems inside of your SelectList.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlListFormElement

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

`public int Size`

If this property is anything higher than 1, which is the default value, the select list will no longer be a 'drop down' type of multiple choices widget, but will show as a 'panel' showing all the choices and making it possible to scroll, if needed, to see more items further down. This property is the number of items that the select list will show at one time. Anything higher than 1, will change its appearance completely and make it into a non-drop-down type of widget.

158. SlidingMenuItem

Magix.UX.Widgets.SlidingMenuItem

Description:

A single sliding menu item. One instance. A SlidingMenu are basically composed out of a whole hierarchy of these types of widgets.

Basetypes

Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The (visible) text of your menu item.

public string AccessKey

The Keyboard Shortcut key for the Widget

`public string URL`

If you give this property a value, the item will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

`public SlidingMenuLevel SlidingMenuLevel`

Will return the SlidingMenuLevel of the MenuItem, if any exists. Every item that have child items, should create a SlidingMenuLevel and put its sub items within the SlidingMenuLevel portion of their markup, or instantiate them within the SlidingMenuLevel child control of those items, if the items are being added programmatically.

`private SlidingMenu SlidingMenu`

Will return the parent SlidingMenu of this item. This property will traverse upwards in the ancestor hierarchy and find the first SlidingMenu instance, being an ancestor of this particular item and return that object.

159. SlidingMenuLevel

Magix.UX.Widgets.SlidingMenuLevel

Description:

Child control of a SlidingMenuItem. Will contain the SlidingMenuItems of a SlidingMenuItem. If you have items which have children, then you must create an item of type SlidingMenuItem and add those children inside of that object. This is true both for .ASPX markup and SlidingMenu hierarchies created in code.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

160. MultiPanelView

Magix.UX.Widgets.MultiPanelView

Description:

Instances of items within a MultiPanel. These are the items a MultiPanel is made of. Most scenarios you would use this control within would have only one of the MultiPanelViews visible at the same time.

Basetypes

Magix.UX.Widgets.Panel

161. MultiPanel

Magix.UX.Widgets.MultiPanel

Description:

A MultiPanel is a collection of MultiPanelViews, often used to create TabControls or similar constructs. Often combined with TabStrip and TabButton. This control can be configured extensively to mimick a lot of different scenarios. Hence we don't call it TabControl, although that happens to be one of the sub-scenarios you can create with it.

Basetypes

Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Methods

public void SetActiveView(int index)

Sets the Active MultiPanelView of this instance according to the index passed in.

public void SetActiveView(MultiPanelView selectedView)

Sets the Active MultiPanelView of this instance according to the MultiPanelView passed in.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

`public int AnimationDuration`

The Speed of the Animation when switching active view.

`public int ActiveMultiPanelViewIndex`

Active/visible view. Will only work if MultiButtonClicked event handler is defined for MultiButton, or you run your own "change logic"...

`public AnimationType AnimationMode`

How to animate AccordionViews when opened, see the AnimationType for details.

162. Panel

Magix.UX.Widgets.Panel

Description:

A container widget for displaying other widgets inside of it. Will render as a div by default, but the specific tag this widget will render, can easily be overridden by changing the Tag property. You can choose to render your panels as paragraphs (p...) for instance. If you only need to display text on your page, and you need to use WebControls for this, you should use the Label control and not the Panel control.

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

Properties

`public string Tag`

The HTML tag element type used to render your widget. You can set this property to anything you wish, including 'address', 'p' or any other types of HTML tags you wish to use to render your widget. If you need an inline-element, such as a span or something, or you need to render a widget without child widgets, you should rather use the Label widget.

`public string DefaultWidget`

The default widget which will be mimicked a 'click' when this widget has focus on the user clicks carriage return (Enter/Return key) on his keyboard. Useful for information typing where you don't want to force the user of clicking a specific 'save' button. If you use another widget as the default widget, you must remember to use the ClientID property of that widget as the DefaultWidget value of this property.

163. RatingControl

Magix.UX.Widgets.RatingControl

Description:

Encapsulates a Rating Control, for allowing the end user to choose number of stars, or 1-x value of something

Basetypes

Magix.UX.Widgets.Panel

Events

public EventHandler Rated

Raised when the user chose to click one of the stars [or other types of objects you are using to signify the value] Use the Value property to retrieve which value the end user gave the object

Properties

public int MaxValue

Maximum number of stars [rating objects]

public int Value

Its value, [0 - MaxValue], 0 equals no-value

`public int Average`

Will show the average score using different colors if you wish

`public bool Enabled`

If true [default] the control is enabled, else disabled

`public bool ShowAlternativeIcons`

If true, will show alternative icons for every second star

164. RadioButton

Magix.UX.Widgets.RadioButton

Description:

A two-state widget type that can be grouped together with other widgets to mimick multiple selections. Like for instance; "Would you like to have coffee, tea or water" is a perfect example of where you would want to use RadioButtons. If you want to group specific radio buttons together, you must give them the same GroupName property. If you do, then only one of these RadioButtons can at any time be 'selected'.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler CheckedChanged

Event raised when the checked state of the widget changes. Use the Checked property to determine if the CheckBox is 'on' or 'off'.

Properties

public string GroupName

If multiple RadioButtons are given the same GroupName, then these RadioButtons will be grouped together, and only one of them can at any time be selected. By playing with this property, you can create multiple groups of radio buttons that will be grouped together accordingly to their GroupName property.

`public bool Checked`

Use this property to determine if the widget is checked or not. If this property is true, then the widget is checked. The default value is 'false'. You can also set this value in your code to change the state of the checked value.

165. TreeItem

Magix.UX.Widgets.TreeItem

Description:

One single item within a TreeView widget. A TreeView is basically nothing but a collection of these types of controls. If you override its CSS class, you can give your TreeItems specific icons and such, in which case you need to change the CSS background-image for those specific MenuItems, which really is out of scope of this article.

Basetypes

Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The (visible) text of your TreeItem.

`public string URL`

If you give this property a value, the item will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

`public bool Expanded`

True if this particular TreeItem is actually expanded.

`public TreeView TreeView`

Will return the parent TreeView of this item. This property will traverse upwards in the ancestor hierarchy and find the first TreeView instance, being an ancestor of this particular item and return that object.

166. Timer

Magix.UX.Widgets.Timer

Description:

Timer widget that will periodically call the server every n'th millisecond. You can set its period through the

Basetypes

Magix.UX.Widgets.Core.BaseControl

Methods

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

protected override string GetClientSideScriptType()

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler Tick
Raised periodically every Duration milliseconds

Properties

public bool Enabled
if true control is enabled and will raise the Tick events every Duration milliseconds, otherwise will not raise tick events before enabled again

public int Interval
Milliseconds bewteen Tick events are raised

167. TreeView

Magix.UX.Widgets.TreeView

Description:

This widget makes it possible for you to create selector kind of widgets that mimicks the representation of a tree kind-of hierarchical structure for displaying things. Useful for displaying for instance folder structures and such.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler SelectedItemChanged

Raised when a new item was selected. Use the SelectedItem property to see which item was actually selected.

Properties

`public TreeItem SelectedItem`
Returns or sets the currently selected item.

168. Window

Magix.UX.Widgets.Window

Description:

Window control. Basically an "advanced panel" with support for moving. It also features borders which can be skinned. The equivalent of a normal desktop window.

Basetypes

Magix.UX.Widgets.Core.CompositeControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler Closed
Raised when window is being closed.

public EventHandler Dragged
Raised when window is moved in browser.

Properties

`public string Caption`

This is the caption of the header parts of the Window.

`public bool Closable`

If this property is true, then the window can be closed by clicking a closing icon. The default value is true.

`public bool Draggable`

If this property is true, then the window can be moved around on the viewport. The default value is true.

`public AspectDraggable Dragger`

Returns the `AspectDraggable`. Be VERY careful with this one, useful for setting boundaries and such, but shouldn't very often need directly editing.

169. TabButton

Magix.UX.Widgets.TabButton

Description:

One instance of an item within a TabStrip. Mostly shown as buttons inside of the TabStrip.

Basetypes

Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The caption of your button. Or the visible text.

public string MultiPanelViewID

MultiPanelView ID associated with this instance. Meaning which MultiPanelView will be visible when button is clicked. If this value is not given, the index of the MultiButtonView will be used as the basis of figuring out which view to expand. Meaning that if this button is the 3rd button in the collection, the 3rd view in the view collection will be expanded.

170. SubMenu

Magix.UX.Widgets.SubMenu

Description:

Child control of a MenuItem. Will contain the SubMenuItems of a MenuItem. If you have MenuItems which have children, then you must create an item of type SubMenu and add those children inside of that object. This is true both for .ASPX markup and Menu hierarchies created in code.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

171. TabStrip

Magix.UX.Widgets.TabStrip

Description:

A strip of TabButtons, used often to trigger changing active view in combination with the MultiPanel control. This control is a collection of TabButtons.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler MultiButtonClicked

Server-side Event Handler raised whan any of the buttons are clicked. If you do not handle this event, then the entire logic will run completely on the client-side, which means it'll run significantly faster due to not having to go to the server at all to do its logic.

Properties

`public ExpansionTrigger ExpansionMode`

What to trigger the opening of the View. Possible values are Click, DbClick and MouseOver.

`public int ActiveMultiButtonViewIndex`

Currently active or visible view's index

`public string MultiPanelID`

ID of MultiPanel associated with this instance. This must be set to something since otherwise you have effectively created a completely useless TabStrip, with no purpose.

172. TextBox

Magix.UX.Widgets.TextBox

Description:

A single-line type of 'give me some text input' type of widget. This widget is a wrapper around the input type="text" type of widget. If you need multiple lines of input, you should rather use the TextArea widget. However this widget is useful for cases when you need the user to give you one line of text input. See also the RichEdit widget if you need rich formatting of your text. This widget can also be set to 'password mode', which means whatever is typed into the widget will not be visible on the screen. Please notice though that by default, this will be transferred to the server in an unsecure manner, so this is only a mechanism to make sure that other people cannot read over your shoulder to see what you're 'secretly' trying to type into your TextBox. Use SSL or other types of security to actually implement safe transmitting of your passwords and similar 'secret text strings'.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Methods

```
public override void RaiseEvent(string name)
```

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

```
protected override string GetEventsRegisterScript()
```

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

`protected override void RenderMuxControl(HtmlBuilder builder)`

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the `HtmlBuilder` class and its related classes.

`protected override void AddAttributes(Element el)`

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as `'el.AddAttribute("attributeName", "attributeValue");'`

Events

`public EventHandler EnterPressed`

If this event is subscribed to, you will get notified when the carriage return, enter or return key is pressed and released. Useful for being able to give 'simple data' without having to physically click some 'save button' to submit it.

Properties

`public TextBoxMode TextMode`

The mode your textbox is set to. The default value of this property is `Normal` which means the characters will display as normal. You can set the mode to `Password`, which will not show the characters as they're written which may be useful for password text boxes.

`public string Placeholder`

Ghost text displayed only if there is no value in the textbox. Useful for giving end user hints and clues about what type of field this is.

`public bool AutoCapitalize`

If true, will automatically Capitalize the first letter of the first word in the textbox, if supported by the browser

`public bool AutoCorrect`

If true, which is the default value, the browser will automatically correct spelling errors and such, which can be very annoying sometimes ...

`public bool AutoComplete`

If true, the default, it will try to automatically complete the form field according to your browser's settings for names and such

`public int MaxLength`

The maximum number of characters this TextBox allows the user to type in.

173. TextArea

Magix.UX.Widgets.TextArea

Description:

A multiple line type of 'give me some text input' type of widget. It wraps the textarea HTML element. If you only need single lines of input, you should probably rather use the TextBox widget. However this widget is useful for cases when you need multiple lines of text input. See also the RichEdit widget if you need rich formatting of your text.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

`public string Placeholder`

Ghost text displayed only if there is no value in the textbox. Useful for giving end user hints and clues about what type of field this is.

174. MenuItem

Magix.UX.Widgets.MenuItem

Description:

Items within a Menu. A MenuItem can have a Text property in addition to a SubMenu, which if defined will popup as a child of the MenuItem. The MenuItem class is basically the individual items of your menu.

Basetypes

Magix.UX.Widgets.Panel

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Text

The Caption of the MenuItem. This is the text displayed within your MenuItem.

public string URL

If you give this property a value, the MenuItem will display as a hyperlink, and the URL property will be what it links to. The Text property will become the anchor text of the URL.

`public SubMenu SubMenu`

Will return the SubMenu of the MenuItem, if any exists. Every MenuItem that have child MenuItems, should create a SubMenu and put its sub MenuItems within the SubMenu portion of their markup, or instantiate them within the SubMenu child control of those MenuItems, if the items are being added programmatically.

`public Menu Menu`

Will return the parent Menu of the MenuItem. This property will traverse upwards in the ancestor hierarchy and find the first Menu instance, being an ancestor of this particular MenuItem and return that object.

175. CheckBox

Magix.UX.Widgets.CheckBox

Description:

A CheckBox is a 'two state button' which you can turn 'on' and 'off'. Useful for boolean UI situations where use must choose between two options, for instance 'yes' or 'no' situations.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

public EventHandler CheckedChanged

Event raised when the checked state of the widget changes. Use the Checked property to determine if the CheckBox is 'on' or 'off'.

Properties

public bool Checked

Use this property to determine if the widget is checked or not. If this property is true, then the widget is checked. The default value is 'false'. You can also set this value in your code to change the state of the checked value.

176. DynamicPanel

Magix.UX.Widgets.DynamicPanel

Description:

Control for making it easier to dynamically instantiate new controls and add them into your page. Completely abstracts away the entire hassle of storing whether or not a control has been previously loaded into the page or not. Make sure you handle the Reload event, and dependent upon the key given will load the exact same control and add it up into the DynamicPanel, and you should experience a very smooth experience in regards to dynamically loading controls into your page. Call the method LoadControl or AppendControl with a unique ID defining which control you wish to load, for instance the name of a UserControl file on disc, and until you explicitly clear your DynamicControl, the same control will be automatically loaded every time. If you do not create a Reload Event Handler for your widget, and you call LoadControl, then the DynamicControl will assume that what you're passing in is a fully qualified path to a UserControl, and attempt to load it as such. You can use the Extra parameter to add extra initialization parameters into the control upon its first load.

Basetypes

Magix.UX.Widgets.Panel

Methods

```
public void LoadControl(string key)
```

Loads a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also AppendControl for having multiple controls within the same DynamicControl. Notice that LoadControl will clear any previously loaded controls from the DynamicControl.

```
public void LoadControl(string key, object extra)
```

Loads a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also AppendControl for having multiple controls within the same DynamicControl. The extra parameter will be passed into the Reload event. Notice that LoadControl will clear any previously loaded controls from the DynamicControl.

```
public void AppendControl(string key)
```

Appends a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also the LoadControl method.

```
public void AppendControl(string key, object extra)
```

Appends a new set of control(s) into the DynamicControl according to the key given. This key will be passed into the Reload event every time onwards until you call ClearControls or leave the page or somehow makes the DynamicControl in-visible or destroy it. The key parameter must be uniquely identifying a specific control type. Make sure you reload the exact same control, every time, in your Reload event handler. If you have not defined a Reload Event Handler, the system will assume you're sending it the complete path and name to a UserControl and attempt to load the given key as a UserControl. See also the LoadControl method. The extra parameter will be passed into the Reload event.

```
public void RemoveFirst()
```

Removes the first Control from the collection of controls, which usually means the oldest control

```
public void ClearControls()
```

Clear the controls and makes sure they won't be (re-)loaded again on the next request.

Events

```
public EventHandler< ReloadEventArgs > Reload
```

Raised when the DynamicControl for some reasons need to load its control(s). Make sure you load the exact same control every time you're given the same Key value in the ReloadEventArgs parameter of your event handler. Also make sure that the Key is uniquely identifying a control such that you cannot get name clashes.

Properties

```
public string Key
```

The entire key for the controls loaded into your DynamicControl. Notice that this might be a list of Keys, separated by pipe (|). Really only exposed for 'advanced functionality'. Do not in general terms use this property for anything. It may be removed in future versions of MUX.

177. ColorPicker

Magix.UX.Widgets.ColorPicker

Description:

ColorPicker widget for allowing the End User to select a Color from a Color Wheel kind of control. Use the ColorWheelImage to override which colors the user is allowed to pick from, if you'd like to restrict him to some sort of palette

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

public bool HasValue()

True if the Widget has any value

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected override string GetClientSideScriptType()

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

Properties

public string ColorWheelImage

Sets which Image to use as the foundation for selecting a Color. By changing this value, you can for instance [kind of] restrict the user in regards to which colors he is allowed to pick from

public Color Value

The selected value of the control

178. AccordionView

Magix.UX.Widgets.AccordionView

Description:

Panels contained inside the Accordion control. Basically an Accordion is only a collection of these types of controls.

Basetypes

Magix.UX.Widgets.Core.CompositeControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Properties

public string Caption

The Header text of the AccordionView. The text displayed at its top.

179. Accordion

Magix.UX.Widgets.Accordion

Description:

Accordions are panels, or `AccordionViews` grouped together where you can expand and collapse individual `AccordionViews` within a group. Normally only one `AccordionView` can be visible at the same time, but this is configurable.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

`public void SetActiveView(int index)`

Will set the active `AccordionView` of the `Accordion` to the specified index.

`public void SetActiveView(AccordionView selectedAccordionView)`

Sets the active `AccordionView` to the given one.

Events

`public EventHandler ActiveAccordionViewChanged`

Event raised when the active `AccordionView` is changed. If you allow multiple `AccordionViews` to be opened at the same time, then this event will not be raised.

Properties

`public int ActiveAccordionViewIndex`

Index of the current Active `AccordionView`. If you allow multiple `AccordionViews` to be open at the same time, then this value doesn't make much sense.

`public AnimationType AnimationMode`

How to animate `AccordionViews` when opened, see the `AnimationType` for details.

`public ExpansionTrigger ExpansionMode`

What to trigger the opening of the View, see the `ExpansionTrigger` for details.

`public int AnimationDuration`

How many milliseconds it will take to animate when an accordion is opened. This property doesn't make sense if you're using the `Accordion` in "NoAnimation" mode.

180. Button

Magix.UX.Widgets.Button

Description:

A clickable button. The equivalent of `input type="button"`. Use when you need a clickable thing to resemble a button. See also the `LinkButton` for an alternative. Also remember that any Widget in Magix UX can be made clickable, so you can also use a `Label` as your 'clickable thingie' if you wish. Even though anything can be made clickable in Magix UX, it is often an advantage to use buttons or link buttons since these elements will mostly be recognized by screen readers and such, and it is hence more 'polite' to use these specially designed types of 'clickable objects' such as the `Button`.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

`protected override void RenderMuxControl(HtmlBuilder builder)`

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the `HtmlBuilder` class and its related classes.

`protected override void SetValue()`

Abstract method you need to implement to receive the value from the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

`protected override void AddAttributes(Element el)`

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as `'el.AddAttribute("attributeName", "attributeValue");'`

181. HiddenField

Magix.UX.Widgets.HiddenField

Description:

Hidden field widget. Useful for having state you wish to pass on to the client but don't want it to be visible for the end user. Notice that this is not a safe place to put things that the user is not supposed to see, like passwords and such. Do NOT trust the value of this element to not be tampered with. Alternatives for using this widget is ViewState, cookies and the Session object. ViewState and cookies are neither 'safe' against tampering.

Basetypes

Magix.UX.Widgets.Core.BaseControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

`public string Value`

The value of the hidden field. The default value is `string.Empty`. Please notice that the `HiddenField` is not a 'safe' place to store things you do not want to show the end user, so only use this for non-secure-critical information, such as for instance UI states and such. No 'secrets' in this value, since this value is very easy to figure out for the end user.

182. LinkButton

Magix.UX.Widgets.LinkButton

Description:

This widget is another type of 'button widget', though this will be rendered using anchor HTML element (anchor element...) Even though everything can be made 'clickable' in Magix UX, it is definitely semantically much more 'correct' to constraint yourself to the ones that are expected to be 'clickable', such as this widget (LinkButton), Button, ImageButton etc. Among other things screen-readers and such will recognize these types of elements as 'clickable' and present the end user with the option of clicking these types of widgets.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

Properties

`public override string Text`

The text property of the LinkButton. This is the 'anchor text' of the widget. Basically what the user will 'see' on the screen.

183. Label

Magix.UX.Widgets.Label

Description:

A 'text widget'. The basic purpose of this widget is purely to display text and nothing else, though through the `CssClass` property and the `Style` property you can easily manipulate this to do mostly anything you wish. If a more 'complex widget' is needed, for instance to host other widgets, then the `Panel` widget is more appropriate to use than the `Label`. Unless the `Tag` property is changed, this widget will render as a `span`...

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the `HtmlBuilder` class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as `'el.AddAttribute("attributeName", "attributeValue");'`

Properties

`public string Text`

The text property of your label. This is what the user will see of your widget.

`public string Tag`

The HTML tag element type used to render your widget. You can set this property to anything you wish, including 'address', 'p' or any other types of HTML tags you wish to use to render your widget. If you need a 'div' HTML element though, or you need to render a widget with child widgets, you should rather use the Panel.

`public string For`

Useful for associating a label with an HTML FORM input element, such as a CheckBox or a RadioButton etc. Notice that this property can only be legally set if the Tag property is of type "label", which is NOT the default value.

184. Menu

Magix.UX.Widgets.Menu

Description:

Menu control. A Menu is basically a collection of MenuItems. Mimicks a Menu the way you're used to seeing them on desktop systems, or at other web applications. The Magix UX menu have support for any number of child menus, and are very customizable in regards to how it looks and behaves.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

Events

public EventHandler LeafMenuItemClicked

Raised when a Leaf MenuItem, meaning a MenuItem with no children have been clicked. Normally these are the MenuItems you need to handle on the server, while the ones with children are basically just 'groupings' of other MenuItems.

public EventHandler MenuItemClicked

Raised when any MenuItem have been clicked.

Properties

`public ExpansionTrigger ExpansionMode`

How to animate items when opened, or what to trigger the opening of the item. Your choices here are Click, DblClick and Hover.

185. Image

Magix.UX.Widgets.Image

Description:

Image Ajax Widget. Useful for showing images that needs Ajax functionality somehow. Notice that most times it's more efficient to display other types of widgets, such as the Panel or a Label and set it to display an image through using something such as background-image through CSS or something similar.

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

`public string ImageUrl`

The URL of where your image is. This is the image that will be displayed to the end user.

`public string AlternateText`

This is the text that will be displayed if the link to the image is broken. It is also the text that most screen-readers will read up loud to the end user.

`public string AccessKey`

What keyboard shortcut the user will have to use to mimick a 'click' on the widget. Often the keyboard shortcut will be mixed up with other keys, depending upon your operating system or browser. For FireFox on Windows for instance the keyboard combination is ALT+SHIFT+whatever key you choose here. So if you choose 'J' as the AccessKey, the user will have to hold down ALT+SHIFT and press 'J' to use the keyboard shortcut.

186. HyperLink

Magix.UX.Widgets.HyperLink

Description:

A wrapper around a hyper link or anchor HTML element (anchor HTML element ...) Sometimes you will need to create links that might change or needs changes after initially created. For such scenarios, this widget is highly useful.

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

public string Text

The anchor text for your hyperlink. This is the text that will be visible in the browser.

`public string URL`

The URL for your link. This is where the user ends up if he clicks your anchor text.

`public string Target`

Which browser window the URL will be opened in. `_blank` will assure it'll always be a new window, while most other names will then only open a new window if there's no window already opened with that name

`public string AccessKey`

What keyboard shortcut the user will have to use to mimick a 'click' on the widget. Often the keyboard shortcut will be mixed up with other keys, depending upon your operating system or browser. For FireFox on Windows for instance the keyboard combination is ALT+SHIFT+whatever key you choose here. So if you choose 'J' as the AccessKey, the user will have to hold down ALT+SHIFT and press 'J' to use the keyboard shortcut.

187. ImageButton

Magix.UX.Widgets.ImageButton

Description:

Technically this widget is completely redundant towards the Image Ajax Widget. But it is here for cases where you want to be semantically highly correct and need something that gives clues in regards to that it is 'clickable'. If you have an Image which is clickable, then semantically this widget is more correct to use.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

protected override void SetValue()

Abstract method you need to implement to receive the value from the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override void RenderMuxControl(HtmlBuilder builder)

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the HtmlBuilder class and its related classes.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Properties

`public string ImageUrl`

The URL of where your image is. This is the image that will be displayed to the end user.

`public string AlternateText`

This is the text that will be displayed if the link to the image is broken. It is also the text that most screen-readers will read up loud to the end user.

188. CompositeControl

Magix.UX.Widgets.Core.CompositeControl

Description:

Abstract helper widget for widget developers. Used internally in e.g. Window and Accordion. Useful for those cases where you have a widget which is 'composed' out of both a normal 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself. The Window is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Caption and a Close button. That's why the Window is a 'CompositeControl'.

Basetypes

Magix.UX.Widgets.Panel

Properties

public Panel Content

The Content area of your widget. This is where you would want to add up child controls to make them a part of the child controls collection of your widget.

public Panel Control

This is your actual 'control area'. This is, among other things, where you want to add up Aspects and such if you want to add aspects to widgets which are CompositeControls. In code you can use 'myControl', but in markup you have to use Control to gain access to the control itself. This is because of that the Content property is the default for your control.

189. BaseWebControlListFormElement

Magix.UX.Widgets.Core.BaseWebControlListFormElement

Description:

Abstract base class for widgets which are HTML FORM or multi-choice type of elements

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElement

Methods

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

Events

`public EventHandler SelectedIndexChanged`

Raised whenever the selected value of your select list is changed. You can also set the selected value in code through using for instance the `SelectedIndex` property. Whenever a user changes the actively selected item of your select list, this event will be raised.

Properties

`public ListItemCollection Items`

A collection of all the items you have inside of your select list. This will be automatically parse through your .ASPX syntax if you declare items inside of the .ASPX file, or you can also programmatically add items in your codebehind file.

`public ListItem SelectedItem`

Will return the currently selected item or set the currently selected item. There are multiple duplicates of this property, like for instance the `SelectedIndex` property. The default `SelectedItem` will always be the first (zero'th) element, regardless of which property you're using to retrieve it.

`public int SelectedIndex`

Will return the index of the currently selected item or set the currently selected item based on its index. There are multiple duplicates of this property, like for instance the `SelectedItem` property. The default `SelectedItem` will always be the first (zero'th) element, regardless of which property you're using to retrieve it. Meaning that the default value of this property will always be '0'.

190. ViewCollectionControl-g

Magix.UX.Widgets.Core.ViewCollectionControl-g

Description:

Abstract base class for widgets which are supposed to contain only one type of child widgets. Contains a lot of usable shorthands like for instance Enumerable support, and an index operator overload.

Basetypes

Magix.UX.Widgets.Panel

Properties

public IEnumerable< T > Views
Enumerable of all the Views inside the control

public T this[int index]
Shorthand to retrieve a specific View

public T this[string ID]
Shorthand to retrieve the view with the given value as its ID

191. CompositeViewCollectionControl-g

Magix.UX.Widgets.Core.CompositeViewCollectionControl-g

Description:

Abstract helper widget for widget developers. Used internally in e.g. TreeItem. Useful for those cases where you have a widget which is 'composed' out of both a normal 'surface', where the user can add up inner child widgets and such, and also 'chrome widgets', or widgets that are a part of the actual widget itself, and you in addition to this also have a specific type of widgets that are supposed to be its child control collections, like for instance the TreeItem are only supposed to have TreeItems within itself. The TreeItem of the TreeView is one such example since it has a 'surface' where the user might add up inner controls, text or such. But it also have a Text property and many other child controls. In addition it is supposed to only have child controls of type TreeItem. That's why the TreeItem is a 'CompositeViewCollectionControl'.

Basetypes

Magix.UX.Widgets.Core.ViewCollectionControl-g

Properties

public Panel Content

The Content area of your widget. This is where you would want to add up child controls to make them a part of the child controls collection of your widget.

public Panel Control

This is your actual 'control area'. This is, among other things, where you want to add up Aspects and such if you want to add aspects to widgets which are CompositeControls. In code you can use 'myControl', but in markup you have to use Control to gain access to the control itself. This is because of that the Content property is the default for your control.

192. BaseWebControlFormElementText

Magix.UX.Widgets.Core.BaseWebControlFormElementText

Description:

Abstract base class for WebControls which are BaseWebControlFormElement type of controls, but also have a Text property. Text property is overridable, its default implementation sets the 'Value' property on the client-side, which may or may not be suitable for your needs.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElement

Properties

public string Text

The text that is displayed within the control, default value is string.Empty. This is very often a string next to the widget which the user can read.

193. BaseWebControl

Magix.UX.Widgets.Core.BaseWebControl

Description:

MUX WebControl equivalent. Contains a couple of additions to the MuxBaseControl. Among others a style attribute and a CssClass. Also contains most of all the DOM event handlers you can possibly handle in Magix UX. In addition to most of the client-side effects you can possibly run on your MUX controls. Class is abstract, and hence cannot be instantiated directly but is meant for inheriting from.

Basetypes

Magix.UX.Widgets.Core.BaseControl

Methods

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

Events

`public EventHandler Click`
Event raised when user clicks your widget.

`public EventHandler DbClick`
Event raised when user double-clicks your widget.

`public EventHandler MouseDown`
Event raised when user pushes his (normally) left mouse button down while the mouse cursor is over your widget. Normally you'd be more interested in handling the Click event. But this event might have its uses too.

`public EventHandler MouseUp`
Event raised when user have pushed his (normally) left mouse button down while the mouse cursor is over your widget, and then releases the mouse button. Normally you'd be more interested in handling the Click event. But this event might have its uses too.

`public EventHandler MouseOver`
Event raised when user moves his mouse over your widget. The equivalent of 'hover'.

`public EventHandler MouseOut`
Event raised when user moves his mouse over your widget, and then moves it away from your widget. The equivalent of 'hover-finished'.

`public EventHandler KeyPress`
Event raised when a character is typed into the widget. Useful for being able to trap typing into e.g. a TextBox or something similar.

`public EventHandler EscKey`
Event raised when the escape key is pressed

Properties

`public string CssClass`
The CSS class(es) of your widget. To explain this is really out of the scope of this library. Pick up any good CSS book to understand how to use this property.

`public string Dir`
The direction of text within your widget. Legal values are 'rtl' and 'ltr'. These means 'right to left' and 'left to right' and signify the direction of your text. Useful for being able to defined reading directions for non-latin based languages such as Hebrew and Arabic.

`public string ToolTip`
The tooltip of your widget. Will display a tiny box with your text whenever the user is hovering his mouse over your widget. There are other and richer ways to create more advanced tooltips in Magix UX, but no other ways will demand less bandwidth and less resources on your user's clients. The HTML attribute 'title' is the value actually sent over to the client when the ToolTip is being used.

`public Effect ClickEffect`
What effect(s) will be ran when the user clicks your widget.

`public Effect DbClickEffect`
What effect(s) will be ran when the user double clicks your widget.

`public Effect MouseOverEffect`

What effect(s) will be ran when the user moves his mouse over your widget.

`public Effect MouseOutEffect`

What effect(s) will be ran when the user moves his mouse out of the surface occupied by your widget on the screen.

`public Effect MouseDownEffect`

What effect(s) will be ran when the user clicks his left mouse button on top of your widget, but before he releases it.

`public Effect MouseUpEffect`

What effect(s) will be ran when the user have clicked your widget, and then release his mouse button. In most cases the the ClickEffect will be more appropriate to use, but this event effect has its uses too.

`public Effect KeyPressEvent`

What effect(s) will be ran when the user have types characters into your widget.

`public Effect EscKeyEffect`

What effect(s) will be ran when the user clicks ESC

`public string TabIndex`

Every DOM element on the client-side has a 'tab order' which defines the order widgets will be given focus while the user is tabbing through your web page. The lower this number is, the earlier in the chain the specific widget will be given focus when tabbing. This is the value defining the tab order.

`public StyleCollection Style`

The inline styles of your widget. An exhaustive explanation of this property is truly out of scope of this file, pick up any good book on the subject CSS to understand how to use this property. But specifically for Magix UX, any style properties added, removed, changed and so on any times during its life-cycle, will automatically, and extremely bandwidth efficient be changed/added/removed Automagically by the core engine of MUX.

194. BaseControl

Magix.UX.Widgets.Core.BaseControl

Description:

Abstract base class for mostly all widgets in Magix UX. This is where the Ajax 'core engine' to a large extent exists within Magix UX. Contains several interesting features, such as for instance the Info property, which can take any string information and serialize back and forth between server-requests. If you need to create your own Ajax Control, you should either directly, or most probably indirectly, inherit from this MuxBaseControl. If you inherit, directly or indirectly, from this class you need to override several methods from this class. The most notable is RenderMuxControl. If you have a 'visual widget', you will mostly inherit from MuxBaseWebControl instead of this class. If your control is 'non-visual', such as the MUX Timer, you will mostly inherit from this class.

Methods

protected void RenderJson(HtmlTextWriter writer)

Only override this one if you truly know what you're doing, and you really need some modified logic. See how the Aspects work for a working example of how this method can be overridden.

protected void SetJsonValue(string key, object value)

This one should be used in properties and such for control developers. Please notice that there is no way we can track values being set before the ViewState is finished loading, and hence all properties and such being set before the ViewState is loaded will be ignored...

protected void SetJsonGeneric(string key, string value)

This method will set a 'generic' property for you control. Useful for JS client-side mappings that doesn't really exist since these key/values pairs will be rendered as additional attributes back to the client.

`protected void SerializeJSONValue(string key, object value, StringBuilder builder)`

Helper method to serialize **ONE** JSON value. Can by default handle a range of different types, such as Rectangle, Point, string, decimal, int, bool etc. If you need to be able to serialize other types, then this can easily be accomplished through overriding this method in your own widget and handle that specific type/property yourself.

`protected string GetClientSideScript()`

Method for registering the control on the client-side. Most control will only need this base functionality, but most controls that have specific JavaScript needs, will need to override this method. One example of a control that needs to override this method is the MUX Timer.

`protected string GetClientSideScriptType()`

If the only thing you need to override when you're creating your own controls are the 'JavaScript type', then you should override this method only to change the JS type being sent back as the registering script to the client. The default value is 'MUX.C', which is a shorthand for 'MUX.Control', which is the common Control JS class for most controls in MUX.

`protected string GetEventsRegisterScript()`

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'], ['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

`protected string GetClientSideScriptOptions()`

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

`protected void RenderMuxControl(HtmlBuilder builder)`

For Ajax Control creators, this is your most important method. This is the method that expects you to build up the HTML markup for your control. Most controls will only build one HTML DOM element, but you can build any amount of complexity up with the help of the `HtmlBuilder` class and its related classes.

`protected void AddAttributes(Element el)`

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as `'el.AddAttribute("attributeName", "attributeValue");'`

`protected string GetControlInvisibleHTML()`

Some, but very few, controls need specific in-visible HTML. The default value of this method will be rendering a `'span'` with the style value of `display:none`. Some custom widgets needs to be able to override this markup, for instance because they needs to render list-elements (HTML `li` element) to be semantically correct. Override this method to make sure your HTML becomes semantically correct for such cases.

`public void ReRender()`

Forces the control to re-render as HTML. This is basically a revert to `UpdatePanel` logic for those extreme cases where such is needed. In general terms you should really only use this method if you either are wrapping non-MUX controls inside of MUX Ajax controls, or you have changed the control collection by either adding or removing new controls from this control's `Control` collection. There exists several tutorials on this matter at the MUX website.

`public void RaiseEvent(string name)`

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

```
public bool AreAncestorsRenderingHTML()
```

Will return true, if for some reasons, any ancestor control are being rendered as pure HTML. This might happen due to an ancestor control are being re-rendered, or it may happen due to an ancestor control being set to visible for the first time. This is in general terms an **EXPENSIVE** method call since it needs to traverse the entire ancestor hierarchy until it hits the Page. Hence be **CAREFUL** with using it...!!

```
public Dictionary( string, string ) GetJsonValues(string key)
```

Retrieves a specific JSON collection values. Used by the style classes and to add up generic attributes to the DOM elements.

```
public override void Focus()
```

If called, this will set the focus to this widget on the client-side. Please notice that there's no 'multiple calls guard' in this logic, which means if you call this method for several controls, only the last method call will actually 'succeed'...

Events

```
public EventHandler InitiallyLoaded
```

Event raised when the Widget is rendered for the first time, meaning it will be triggered upon re-renderings and such, but not during normal postbacks. But obviously, if a parent Widget is being re-rendered, then also this Event will be raised again, even though the Widget hasn't changed any of its properties.

Properties

`public string Info`

Additional information you wish to attach to your widget somehow. Useful for things such as Repeaters where you have the same event handler for several buttons on your page, but need to know specifically which button was actually pressed. PS! Just like the HiddenField the value in this property should not be trusted to be securely serialized since internally it uses the ViewState to serialize this additional information, and ViewState is (by default) not encrypted in any ways.

`public bool HasRendered`

Will return true if this widget has been rendered before. Kind of like an 'IsPostBack logic' on a control level. Useful for determining if this is the first time this control is being rendered, or not.

`public bool RenderingHtml`

Returns true if this control is either not rendered from before, or has been signaled to re-render as HTML (and JS object registration) Notice though that the control still might be rendering HTML due to an ancestor control, any way upwards in this control's ancestor hierarchy has been signaled to re-render. Meaning, this is not by any means any 'absolute definition' of whether or not this control will render HTML. If you need absolute proof for some reasons of whether or not this control will render as HTML you need to check this property, but also the AreAncestorsRenderingHTML method to find out whether or not any of its ancestor controls are re-rendering.

`public IEnumerable< AspectBase > Aspects`

Iterator for enumerating all aspects within this specific widget.

195. BaseWebControlFormElementInputText

Magix.UX.Widgets.Core.BaseWebControlFormElementInputText

Description:

Abstract base class for widgets being BaseWebControlFormElementText type of controls, but also uses the Text portions as an input value which the user can change himself through interacting with the widget.

Basetypes

Magix.UX.Widgets.Core.BaseWebControlFormElementText

Methods

public void Select()

Selects the whole text portions of the widget and gives the widget focus. Useful for widgets where you know the user will want to change its entire Text content when returned.

public override void RaiseEvent(string name)

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

protected override void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetClientSideScriptOptions()

This method makes it possible for you to return custom options according to what properties/options your custom widget have. The method is expected to return something similar to "handle:'xyz',color:'Red'" which will automatically map towards the JS client-side options of your custom widget class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'

Events

public EventHandler TextChanged

Raised when widget changes its Text property. Normally a TextBox and a TextArea won't raise this event before they're loosing focus, since that's when we know what the Text property is actually changed to.

public EventHandler EscPressed

Raise when widget has focus and user clicks ESC. Useful for being able to create 'discardable operations' types of logic.

196. BaseWebControlFormElement

Magix.UX.Widgets.Core.BaseWebControlFormElement

Description:

Abstract base class for widgets which are HTML FORM type of elements. Abstracts away things such as blur and focus event handling, enabled, keyboard shortcuts and other internals things. Inherit from this one if you intend to create an Ajax Widget which wraps an HTML FORM element.

Basetypes

Magix.UX.Widgets.Core.BaseWebControl

Methods

protected abstract void SetValue()

Abstract method you need to implement to receive the value form the client and serialize it into the server-side web control. Override this one and extract the value of the control from the HTTP Request if you inherit from this class.

protected override string GetEventsRegisterScript()

This is the method you need to override, when creating your own controls, and you have server-side events you need to handle. The return value of this method is expected to be something such as "['click'],['mouseover']" etc. It will create handlers for those DOM events on the client-side and automatically create an Ajax Request going towards the server when those DOM events are raised.

protected override void AddAttributes(Element el)

Override this method to add up attributes and their values to your custom widget. Only override this method if you're an Ajax Custom Control builder. Within this method you're expected to only put logic such as 'el.AddAttribute("attributeName", "attributeValue");'


```
public override void RaiseEvent(string name)
```

Override this method to handle events that are being raised on the client-side. Theoretically you can raise events in code yourself to mimick whatever event you wish to mimick, but 99% of the times you think you would want to do such a thing, you're really doing something very bad. DON'T...! At least unless you know very well that this is exactly what you want to do.

Events

```
public EventHandler Blur
```

Event being raised when the widget does no longer have focus. The exact opposite of the Focused event. Will trigger whenever the widget loses focus for some reasons. Notice that the widget (obviously) needs to **have** focus before it can 'lose' it. Some ways a widget may lose focus is when the user tabs through the widgets on the screen. Another way is clicking another widget, and thereby loosing focus for the previously focused widget.

```
public EventHandler Focused
```

The Focus event. The exact opposite of the Blur event. This one will be raised whenever the widget gains focus. Gaining focus can happen several ways, first of all the user might click the specific widget, which will give that widget focus. Secondly the user might tab through the widgets until the specific widget is reached. When the widget gains focus, this event will be raised.

Properties

```
public string AccessKey
```

What keyboard shortcut the user will have to use to mimick a 'click' on the widget. Often the keyboard shortcut will be mixed up with other keys, depending upon your operating system or browser. For FireFox on Windows for instance the keyboard combination is ALT+SHIFT+whatever key you choose here. So if you choose 'J' as the AccessKey, the user will have to hold down ALT+SHIFT and press 'J' to use the keyboard shortcut.

public bool Enabled

Boolean value indicating whether or not the widget is enabled. If this one is false, then the user cannot change the value of the widget by interacting normally with it. Please note that this is not safe and can be overridden by tools and manipulating the HTTP stream directly and so on. Do not trust values from controls that are disabled to be the value you explicitly gave them earlier.

public Effect BlurEffect

What Effect will run when user is making the widget lose focus.

public Effect FocusedEffect

What Effect will run when user is making the widget gain focus.