WASP AT - Wasp Analysis Tools

WASP Builder
Routine Descriptions

Developed by:

Integrated Decision Support Group
The Water Center
Colorado State University
Fort Collins, Colorado

Developed for:

U.S Department of the Interior
Bureau of Reclamation
Technical Services Center
Denver, Colorado
# Table of Contents

## 1.0 C++ Classes for Wasp Builder by Source File

<table>
<thead>
<tr>
<th>File Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BypassOptsDlg.h</td>
<td>1</td>
</tr>
<tr>
<td>CardCompleteDlg.h</td>
<td>1</td>
</tr>
<tr>
<td>CntrItem.h</td>
<td>2</td>
</tr>
<tr>
<td>DeleteItemsDlg.h</td>
<td>4</td>
</tr>
<tr>
<td>exchangenodedlg.h</td>
<td>4</td>
</tr>
<tr>
<td>GraphControlDlg.h</td>
<td>5</td>
</tr>
<tr>
<td>GraphDlg.h</td>
<td>5</td>
</tr>
<tr>
<td>GroupADlg.h</td>
<td>12</td>
</tr>
<tr>
<td>GroupBDlg.h</td>
<td>13</td>
</tr>
<tr>
<td>groupbnodedlg.h</td>
<td>14</td>
</tr>
<tr>
<td>GroupCDlg.h</td>
<td>15</td>
</tr>
<tr>
<td>groupcnodedlg.h</td>
<td>16</td>
</tr>
<tr>
<td>GroupDDlg.h</td>
<td>17</td>
</tr>
<tr>
<td>groupdnodeldlg.h</td>
<td>18</td>
</tr>
<tr>
<td>GroupEDlg.h</td>
<td>20</td>
</tr>
<tr>
<td>groupenodedlg.h</td>
<td>21</td>
</tr>
<tr>
<td>GroupFDlg.h</td>
<td>22</td>
</tr>
<tr>
<td>GroupFNodeDlg.h</td>
<td>24</td>
</tr>
<tr>
<td>GroupGDlg.h</td>
<td>25</td>
</tr>
<tr>
<td>groupgnodedlg.h</td>
<td>26</td>
</tr>
<tr>
<td>GroupHDlg.h</td>
<td>27</td>
</tr>
<tr>
<td>GroupIDlg.h</td>
<td>29</td>
</tr>
<tr>
<td>GroupJDlg.h</td>
<td>30</td>
</tr>
<tr>
<td>InputPlotBDlg.h</td>
<td>31</td>
</tr>
<tr>
<td>InputPlotCDlg.h</td>
<td>32</td>
</tr>
<tr>
<td>InputPlotDDlg.h</td>
<td>32</td>
</tr>
<tr>
<td>InputPlotEDlg.h</td>
<td>32</td>
</tr>
<tr>
<td>InputPlotFDlg.h</td>
<td>32</td>
</tr>
<tr>
<td>InputPlotIDlg.h</td>
<td>33</td>
</tr>
<tr>
<td>labelnodedlg.h</td>
<td>33</td>
</tr>
<tr>
<td>MultiTrack.h</td>
<td>34</td>
</tr>
</tbody>
</table>
WASP Builder Routine Description

OutputDlg.h ................................................................. 34
PaletteDlg.h ............................................................... 35
ProjectEditDlg.h ......................................................... 35
RangeDlg.h ................................................................. 35
RCSClass.h ................................................................. 36
RCSOpenDlg.h ............................................................. 37
RCSSaveDlg.h .............................................................. 37
ReportData.h ............................................................... 38
SegmentChooserDlg.h .................................................... 43
SegmentInsertDlg.h ...................................................... 43
SegmentNodeDlg.h ....................................................... 43
SensAnalysisParamsDlg.h ............................................... 44
SensitivityDlgB.h ......................................................... 44
SensitivityDlgC.h ......................................................... 45
SensitivityDlgD.h ......................................................... 45
SensitivityDlgE.h ......................................................... 46
SensitivityDlgF.h ......................................................... 46
SensitivityDlgG.h ......................................................... 47
SensitivityDlgH.h ......................................................... 47
SensitivityDlgJ.h ......................................................... 48
SimTypeDlg.h .............................................................. 48
TimeStepDlg.h .............................................................. 48
WaspBuilder.h ............................................................ 49
WaspBuilderDoc.h ......................................................... 49
WaspBuilderView.h ...................................................... 56
WaspClass.h ............................................................... 58
WASPReportDoc.h ......................................................... 58
waspreportview.h ......................................................... 60
WaspUtils.h ............................................................... 60

2.0 Function Index Arranged by Class ........................................ 63
2.0 Member Index Arranged by Class ....................................... 69
1.0 C++ Classes for Wasp Builder by Source File

1. Source File: BypassOptsDlg.h

Class: BypassOptsDlg
Function: BypassOptsDlg(StringArray &bypassOptsArrayList, char **systemLabels, CWnd* pParent = NULL) : Constructor
Description: A checkbox dialog used for setting WASP bypass options.
Input Args: bypassOptsArrayList: A string list of "0"s (off) or "1"s (on). Each index corresponds to the same index in systemLabels.
Input Args: systemLabels: An array with the same count as bypassOptsArrayList, usually WaspSystemLabels[simtype].
Input Args: pParent: the parent window.

Class: BypassOptsDlg
Member: m_bypassOptsArrayList : StringArray
Description: Used to initialize m_systemCheckList. Contains the user's changes when OK in dialog is pressed; unchanged if dialog cancelled.

Class: BypassOptsDlg
Member: m_systemCheckList : CChekListBox
Description: The check box list.

2. Source File: CardCompleteDlg.h

Class: CardCompleteDlg
Function: CardCompleteDlg(CWaspBuilderDoc *pDoc, CWnd* pParent = NULL) : Constructor
Description: A dialog showing data deficiencies in user supplied input. Helpful to see what may cause problems in the model before running it.
Input Args: pDoc: needed in order to get network card data.
Input Args: pParent: the parent window.

Class: CardCompleteDlg
Function: UpdateAll() : void
Description: Refresh the dialog contents.
3. Source File: CntrItem.h

Class: CWaspBuilderCntrItem
Function: CWaspBuilderCntrItem(CWaspBuilderDoc* pContainer) : Constructor
Description: Controls the display of the nodes that appear on the network view. Contains subdialogs that act as storage for the node card data.
Input Args: pDoc: needed in order to get global card data.

Class: CWaspBuilderCntrItem
Function: Invalidate(CView* pNotThisView = NULL) : void
Description: Forces redraw of this node in all views attached to the document except pNotThisView.
Input Args: pNotThisView: don't send an update message to this view.

Class: CWaspBuilderCntrItem
Function: Draw(CDC* pdc, const CRect& rcBounds) : void
Description: Renders the node based on m_nodeType
Input Args: pdc: Rendering context.
Input Args: rcBounds: Area to draw in.

Class: CWaspBuilderCntrItem
Function: Move(CRect &rc) : void
Description: Moves the physical location of the node to another part of the view.
Input Args: rc: The new location.

Class: CWaspBuilderCntrItem
Function: GetSize() : CSize
Description: Returns the bounds of the node.
Returns: The extent of the node.

Class: CWaspBuilderCntrItem
Function: SetSize(CSize size) : void
Description: Sets the new bounds of the node.
Returns: The new extent of the node.

Class: CWaspBuilderCntrItem
Function: GetRect() : CRect
Description: Gets the bounds and placement of the node on the view.
Returns: The extent and location of the node.

Class: CWaspBuilderCntrItem
Function: SetRect() : void
Description: Sets the bounds and placement of the node on the view.
Returns: The new extent and location of the node.

Class: CWaspBuilderCntrItem
Function: GetMidPoint() : CPoint
Description: Gets the midpoint of the node relative to the entire view.
Returns: The midpoint of the node on the view.
1.0 C++ Classes for Wasp Builder by Source File

Class: CWaspBuilderCntrolItem
Function: UpdateExtent() : BOOL
Description: Redraws the node. Sets the extent of the node if uninitialized based on node type.
Returns: TRUE (never fails).

Class: CWaspBuilderCntrolItem
Function: SetNodeType(NodeEnum new_type) : void
Description: Assigns m_nodeType. Creates data structures needed to store data for the new node type.
Input Args: new_type: the new node type (exchange, label, etc).

Class: CWaspBuilderCntrolItem
Function: GetNodeType() : NodeEnum
Description: Retrieves m_nodeType.
Returns: the type of the node (segment, exchange, etc).

Class: CWaspBuilderCntrolItem
Function: SetParameter(int key, CString val) : void
Description: Sets the parameter represented by 'key' to 'val'. Strings are converted to the parameter data type. No error checking is done; be sure that the parameter is consistent with the node type (i.e. don't pass LABELSTRING to a segment node).
Input Args: key: one of the enums defined in Globals.h
Input Args: val: the new value of the parameter.

Class: CWaspBuilderCntrolItem
Function: GetParameter(int key) : CString
Description: Gets the parameter represented by 'key'. The result is converted to a string. No error checking is done; be sure that the parameter is consistent with the node type (i.e. don't pass LABELSTRING to a segment node).
Input Args: key: one of the enums defined in Globals.h
Returns: The value of the parameter.

Class: CWaspBuilderCntrolItem
Function: SegmentJuxtapose(CWaspBuilderCntrolItem *pSwitchItem) : void
Description: Juxtaposes the locations of two nodes. Useful when nodes are being deleted or renamed, incurring a resharfe/reorder event.
Input Args: pSwitchItem: The node to switch with.

Class: CWaspBuilderCntrolItem
Function: PopupActionWindow() : void
Description: Displays data assignment window.

Class: CWaspBuilderCntrolItem
Function: ParameterChanged(int key) : BOOL
Description: Test if a parameter given by 'key' has been altered by the user. This is useful for determining when a control has made a change that needs to be reflected by the project dialogs.
Input Args: the parameter, one of the enums defined in Globals.h
Returns: TRUE if the parameter has been changed (usually by the node control).
Class: CWaspBuilderCntrItem
Function: Anchor(CWaspBuilderCntrItem* anchornode) : void
Description: Anchor 'anchornode' to 'this' such that whenever 'this' is moved, 'anchornode' moves with it.
Input Args: anchornode: the node that will be attached to this node.

Class: CWaspBuilderCntrItem
Function: Delete() : void
Description: Remove 'this' from the document's list of nodes.

Class: CWaspBuilderCntrItem
Description: Check if the card data for this card is complete, and if not, what the deficiencies are.
Returns: Empty if no deficiencies; otherwise a string containing a warning or error for the first deficiency found.

Class: CWaspBuilderCntrItem
Function: Serialize(CArchive& ar) : void
Description: Stores/Retrieves the node's data from the file object.
Input Args: ar: the file object.

4. Source File: DeleteItemsDlg.h

Class: DeleteItemsDlg
Function: DeleteItemsDlg(CWnd* pParent = NULL) : Constructor
Description: Dialog for choosing items from a multiple selection list. If OKed, m_selectionArray will be populated in reverse order with the selected indices (makes list deletion simpler).

Class: DeleteItemsDlg
Function: SetData(CArray<CString, CString&> &items) : void
Description: Fills m_itemArray, which will be used to initialize the choice list.
Input Args: items: Strings to populate the listbox.

Class: DeleteItemsDlg
Function: SetData(CArray<int, int> &items) : void
Description: Fills m_itemArray, which will be used to initialize the choice list. The integers are converted to strings.
Input Args: items: Integers to populate the listbox.

5. Source File: exchangenodedlg.h

Class: ExchangeNodeDlg
Function: ExchangeNodeDlg(CWaspBuilderDoc* pContainer, CWnd* pParent = NULL) : Constructor
Description: Stores information related to exchange nodes. See CntrItem for additional information.
1.0 C++ Classes for Wasp Builder by Source File

Class: ExchangeNodeDlg
Function: GetChangedParametersArray(CStringList &retList) : void
Description: Retrieves parameters related to exchange nodes that have been changed, allowing the document to update data dependent on this node.
Input Args: retList: A list of key, value pairs.

Class: ExchangeNodeDlg
Function: SetParameter(int key, LPCSTR paramStr) : void
Description: Parameter 'key' is assigned value 'value'.
Input Args: key: the parameter to assign.
Input Args: value: the parameter's new value.

Class: ExchangeNodeDlg
Function: GetParameter(int key) : CString
Description: Retrieve value of parameter 'key'.
Input Args: key: the parameter to query.
Returns: value: the parameter's value.

Class: ExchangeNodeDlg
Function: Serialize(CArchive& ar) : void
Description: Stores/Retrieves the dialog's data from file object.
Input Args: ar: the file object.

Class: ExchangeNodeDlg
Member: m_changedParameters : StringArray
Description: OnOK will update the string of changed parameters.

6. Source File: GraphControlDlg.h

Class: GraphControlDlg
Function: GraphControlDlg(CList<GraphDlg*, GraphDlg*> &graphList, CWnd* pParent = NULL) : Constructor
Description: Dialog for controlling a list of GraphDlg's.
Input Args: graphList: list of pointers of GraphDlg's that are to be controlled.
Returns: optional parent window.

Class: GraphControlDlg
Function: GraphControlDlg() : Destructor
Description: Closes all associated GraphDlg's.

7. Source File: GraphDlg.h

Class: GraphDlg
Function: ~GraphDlg(CWnd* pParent = NULL) : Constructor
Description: Front-end tp CGraph class.
Class: GraphDlg
Function: ~GraphDlg() : Destructor
Description: Frees dynamically allocated memory.

Class: GraphDlg
Function: SetDimensions(int nframes, int nsets, int ncols, int nlabels, int missingVal=0, BOOL useLabelSkip=TRUE, BOOL useDistData=FALSE) : void
Description: Assigns the m_nSets, m_nCols, and m_nLabels variables and allocates memory.
Input Args: nframes: The number of frames that will be allocated. Each frame is a slide that is displayed after a click event. Also can be induced from GraphControlDlg.
Input Args: nsets: The number of sets that will be graphed. Each set represents a group of data and can be assigned a legend title, color, etc, using other routines.
Input Args: ncols: The number of X data points per set.
Input Args: nlabels: The number of X data labels that should be allocated.
Input Args: missingVal: The initial value to be assigned to m_setMissing (0 means point should be displayed, 128 means point is missing).
Input Args: useLabelSkip: if TRUE, then try to come up with a reasonable LabelEvery value so that column labels won't be too squished together.
Input Args: useDistData: Allocate additional data structures (m_setDist) to allow for distance data to be inputted (allows for arbitrary X points).

Class: GraphDlg
Function: SetTitles() : void
Description: Assigns the graph's title, x and y titles, and a title for the dialog window.
Input Args: title: the graph title (m_graphTitle)
Input Args: xtitle: the X axis title (m_XTitle)
Input Args: ytitle: the Y axis title (m_YTitle)
Input Args: windowTitle: the window dialog title (m_windowTitle)

Class: GraphDlg
Function: DrawGraph() : void
Description: Set the graph class data from the GraphDlg data and draw.

Class: GraphDlg
Function: SetFrameData() : void
Description: Load the current frame's data into the graph class and display.

Class: GraphDlg
Member: m_size : CSize
Description: Sets the size of the dialog window. Assign before callingDoModal (unused if Create() is used to instantiate the dialog).

Class: GraphDlg
Member: m_graphType : CGraph::enumGraphType
Description: Sets the graph type (see cgraph.h)
1.0 C++ Classes for Wasp Builder by Source File

Class: GraphDlg
Member: m_graphStyle : CGraph::enumGraphStyle
Description: Sets the graph style (see cgraph.h)

Class: GraphDlg
Member: m_linePattern : int *
Description: If non-Null, then sets CGraph's PatternData (see cgraph.h)

Class: GraphDlg
Member: m_colors : int *
Description: If non-Null, then sets CGraph's ColorData (see cgraph.h)

Class: GraphDlg
Member: m_overlayColors : int *
Description: If non-Null, then sets CGraph's OverlayColor (see cgraph.h)

Class: GraphDlg
Member: m_overlayColors : int
Description: Sets CGraph's ThickLines (see cgraph.h)

Class: GraphDlg
Member: m_currentFrame : int
Description: SetFrameData() uses this member to index the current frame data.

Class: GraphDlg
Member: m_nFrames : int
Description: The number of frames of data. Each frame consists of a complete graph's worth of data.

Class: GraphDlg
Member: m_nSets : int
Description: The number of sets of data. Each set is a data plot on the graph.

Class: GraphDlg
Member: m_setData : double***
Description: The Y data values. Index by [iframe][iset][icol]

Class: GraphDlg
Member: m_setDist : double***
Description: Optional X data values. Index by [iframe][iset][icol]. Only read if m_useDistData set to TRUE.

Class: GraphDlg
Member: m_setDist : double***
Description: Optional missing data values. Index by [iframe][iset][icol]. Always read but defaults to 0 (not missing).

Class: GraphDlg
Member: m_setTitle : CString*
Description: Graph title for each frame.
Class: **GraphDlg**
Member: `m_useDistData : BOOL`
Description: If true, then allocate space for the distance array and assign it to CGraph.

Class: **GraphDlg**
Member: `m_overlayGraphType : int`
Description: Assign to CGraph::OverlayGraphType. Only used if `m_nOverlaySets > 0`.

Class: **GraphDlg**
Member: `m_overlayGraphStyle : int`
Description: Assign to CGraph::OverlayGraphStyle. Only used if `m_nOverlaySets > 0`.

Class: **GraphDlg**
Member: `m_nOverlaySets : int`
Description: Assign to CGraph::OverlayNumSets. Note that the calling routine must handle all data allocs for `m_overlaySetData` and `m_overlaySetDist`, but the destructor will free any memory.

Class: **GraphDlg**
Member: `m_overlaySetData : double***`
Description: Assign to CGraph::OverlayData (Y data points). Note that the calling routine must allocate the memory for this memory, but the destructor will free it when the graph is closed. Index by `[iframe][iset][icol]`.

Class: **GraphDlg**
Member: `m_overlaySetData : double***`
Description: Assign to CGraph::OverlayXPosData, optional X data points. Note that the calling routine must allocate the memory for this memory, but the destructor will free it when the graph is closed. Index by `[iframe][iset][icol]`.

Class: **GraphDlg**
Member: `m_overlaySetMissing : int***`
Description: Assign to CGraph::OverlayExtraData, optional missing data points. Note that the calling routine must allocate the memory for this memory, but the destructor will free it when the graph is closed. Index by `[iframe][iset][icol]`.

Class: **GraphDlg**
Member: `m_overlayTitle : CString`
Description: Assign to CGraph::RightTitle. Note this is always used even no overlay data are defined. Currently implementation uses one right title for every frame.

Class: **GraphDlg**
Member: `m_overlayTitleStyle : int`
Description: Assign to CGraph::RightTitleStyle. Note this is always used even no overlay data are defined.

Class: **GraphDlg**
Member: `m_overlaySetTitle : CString*`
Description: Appended to CGraph::LegendText (after the regular data titles have been assigned).
Class:  
        GraphDlg
Member:  m_useOverlayDistData : BOOL
Description:  If true, thenDrawGraph will assign m_overlaySetDist to CGraph::OverlayXPosData.

Class:  
        GraphDlg
Member:  m_nCols : int
Description:  Number of X points.

Class:  
        GraphDlg
Member:  m_nLabels : int
Description:  If non-zero, SetDimensions will allocated memory for m_labels.

Class:  
        GraphDlg
Member:  m_labels : CString*
Description:  Assigns to CGraph::LabelText if non-NULL.

Class:  
        GraphDlg
Member:  m_dataLabels : CString***
Description:  By allocating memory to this member, SetFrameData will assign
            CGraph::DataLabelText. Index using [iframe][iset][icol]. The destructor will take care
            of freeing the memory.

Class:  
        GraphDlg
Member:  m_XAxisTicks : int
Description:  Assigns to CGraph::XAxisTicks

Class:  
        GraphDlg
Member:  m_XAxisTicksMinor : int
Description:  Assigns to CGraph::XAxisMinorTicks

Class:  
        GraphDlg
Member:  m_labelXDateStart : CString
Description:  Assigns to CGraph::LabelXDateStart if non-empty. Note that currently
            CGraph::LabelXDateInc will be set to "0000:01:01" and CGraph::LabelXFormat will
            be "mmm". CGraph::LabelXType is 1.

Class:  
        GraphDlg
Member:  m_showYAxisTicksLeft : BOOL
Description:  If TRUE, then m_YAxisTicksLeft will be assigned to
            CGraph::YAxisTicks[CGraph::yAxisLeft].

Class:  
        GraphDlg
Member:  m_YAxisTicksLeft : int
Description:  The number of left Y axis ticks to use. This will also be the dimension of
            m_YAxisTextLeft is Y text labels are used.

Class:  
        GraphDlg
Member:  m_YAxisTicksMinorLeft : int
Description:  Assigns to CGraph::YAxisMinorTicks[CGraph::yAxisLeft]. Note that the sign will be
            reversed (not sure why this needs to be so). Defaults to 0.
Class: GraphDlg
Member: m_showYAxisTicksRight : int
Description: If TRUE, then m_YAxisTicksRight will be assigned to CGraph::YAxisTicks[CGraph::yAxisRight].

Class: GraphDlg
Member: m_YAxisTicksRight : int
Description: Assigns to CGraph::YAxisTicks[CGraph::yAxisRight]. Defaults to 0.

Class: GraphDlg
Member: m_YAxisTicksMinorRight : int
Description: Assigns to CGraph::YAxisMinorTicks[CGraph::yAxisRight]. Note that the sign will be reversed (not sure why this needs to be so). Defaults to 0.

Class: GraphDlg
Member: m_YAxisTextLeft : CString*
Description: Assigns to CGraph::YLabelText[CGraph::yAxisLeft] if non-NULL. The dimension must be equal to m_YAxisTicksLeft

Class: GraphDlg
Member: m_YAxisTextRight : CString*
Description: Assigns to CGraph::YLabelText[CGraph::yAxisRight] if non-NULL. The dimension must be equal to m_YAxisTicksRight

Class: GraphDlg
Member: m_windowTitle : CString*
Description: The window title that should be displayed for the current frame. There must be one for each frame.

Class: GraphDlg
Member: m_graphTitle : CString*
Description: The graph title that should be displayed for the current frame. There must be one for each frame.

Class: GraphDlg
Member: m_XTitle : CString*
Description: The X axis title that should be displayed for the current frame. There must be one for each frame.

Class: GraphDlg
Member: m_YTitle : CString*
Description: The left Y axis title that should be displayed for the current frame. There must be one for each frame.

Class: GraphDlg
Member: m_backgroundColor : CGraph::enumColor
Description: The background color of the graph.

Class: GraphDlg
Member: m_legendPos : CGraph::enumLegendPos
Description: Assigns to CGraph::LegendPos. Defaults to bottom of screen.
1.0 C++ Classes for Wasp Builder by Source File

Class: GraphDlg
Member: m_xAxisMin : double
Description: Assigns to CGraph::XAxisMin. Only used if > -10000. Also causes CGraph to use XAxisStyle = CGraph::userDefined

Class: GraphDlg
Member: m_xAxisMax : double
Description: Assigns to CGraph::XAxisMax. Only used if > -10000. Also causes CGraph to use XAxisStyle = CGraph::userDefined.

Class: GraphDlg
Member: m_yAxisMinLeft : double
Description: Assigns to CGraph::YAxisMin[CGraph::yAxisLeft]. Only used if > -10000. Also causes CGraph to use YAxisStyle[CGraph::yAxisLeft] = CGraph::userDefined.

Class: GraphDlg
Member: m_yAxisMaxLeft : double
Description: Assigns to CGraph::YAxisMax[CGraph::yAxisLeft]. Only used if > -10000. Also causes CGraph to use YAxisStyle[CGraph::yAxisLeft] = CGraph::userDefined.

Class: GraphDlg
Member: m_yAxisMinRight : double
Description: Assigns to CGraph::YAxisMin[CGraph::yAxisRight]. Only used if > -10000. Also causes CGraph to use YAxisStyle[CGraph::yAxisRight] = CGraph::userDefined.

Class: GraphDlg
Member: m_yAxisMaxRight : double
Description: Assigns to CGraph::YAxisMax[CGraph::yAxisRight]. Only used if > -10000. Also causes CGraph to use YAxisStyle[CGraph::yAxisRight] = CGraph::userDefined.

Class: GraphDlg
Member: m_useGridX : BOOL
Description: Adds CGraph::vertical to CGraph::GridStyle. Displays a vertical grid at every X tick.

Class: GraphDlg
Member: m_useGridY : BOOL
Description: Adds CGraph::horizontal to CGraph::GridStyle. Displays a horizontal grid at every Y tick.

Member: m_doNotDelete : BOOL
Description: If FALSE, then the dialog will delete itself when closed. Otherwise it will wait for the calling program to do so (used when GraphControlDlg is driving the graphs).

Class: GraphDlg
Function: StartAnimation(int elapseVal) : void
Description: Installs the timeout proc and starts timeseries graphing.
Input Args: elapseVal: the time in milliseconds to advance each frame.
Class: **GraphDlg**
Function: AdvanceFrame() : void
Description: Called to advance the frame manually instead of using a mouse click or timer. Uses m_currentFrame.

8. Source File: **GroupADlg.h**

Class: **GroupADlg**
Function: GroupADlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group A card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: *prjDlg*: backpointer so info from other group dialogs can be accessed.

Class: **GroupADlg**
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card A data from WaspClass.
Input Args: *wc*: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: **GroupADlg**
Function: GetMaxNoSys() : int
Description: Return the maximum number of systems for the current simulation type.
Returns: The maximum number of systems for the current simulation type.

Class: **GroupADlg**
Function: WriteInput(FILE *fp, CString title="", CString title2="") : BOOL
Description: Write card A as valid WASP input file.
Input Args: *title*: Optional TITLE1 to substitute for m_title1.
Input Args: *title2*: Optional TITLE2 to substitute for m_title2.
Returns: TRUE if successful.

Class: **GroupADlg**
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: *ar*: the class data.
Input Args: *version*: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: **GroupADlg**
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: *ar*: the class data.
Returns: TRUE if successful.
1.0 **C++ Classes for Wasp Builder by Source File**

**Class:** GroupADlg  
Function: UpdateJMASS() : void  
Description: Initialize system list for mass balance analysis choice and system choices.

**Class:** GroupADlg  
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void  
Description: Return a list of possible holes in global data.  
Input Args: linelist: output variable that will contain the results.

**Class:** GroupADlg  
Member: m_timestepArrayList StringArray  
Description: Internal list of timesteps.

**Class:** GroupADlg  
Member: m_printIntervalArrayList StringArray  
Description: Internal list of print intervals.

**Class:** GroupADlg  
Member: m_systemBypassArrayList StringArray  
Description: Internal list of bypass options.

**Class:** GroupADlg  
Member: m_isegoutArrayList StringArray  
Description: Internal list of output segments.

9. **Source File:** GroupBDlg.h

**Class:** GroupBDlg  
Function: GroupBDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor  
Description: Interface for the global group B card (ie, displays card data that doesn't include most individual segment or node data.  
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

**Class:** GroupBDlg  
Function: LoadFromWC(WaspClass *wc) : BOOL  
Description: Initialize by reading card B data from WaspClass.  
Input Args: wc: WaspClass created by importing a WASP input data file.  
Returns: TRUE if successful.

**Class:** GroupBDlg  
Function: ReadAR(CArchive& ar, int version) : BOOL  
Description: Update state from archive data.  
Input Args: ar: the class data.  
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.  
Returns: TRUE if successful.
Class: GroupBDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupBDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: GroupBDlg
Member: m_surfwaterArrayList StringArray
Description: Internal 2D list of surface water time function parameters X Number of exchange time functions. Updated only when dialog is OKed.

Class: GroupBDlg
Member: m_porewaterArrayList StringArray
Description: Internal 2D list of porewater time function parameters X Number of exchange time functions. Updated only when dialog is OKed.

Class: GroupBDlg
Member: m_rbykArrayList StringArray
Description: Internal list of exchange bypass options

10. Source File: groupbnodedlg.h

Class: GroupBNodeDlg
Function: GroupBNodeDlg(CWnd* pParent = NULL) : Constructor
Description: Displays group B exchange data dialog.

Class: GroupBNodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to assign to (see GroupB Enums in globals.h).
Input Args: paramStr: CString typed value to assign.

Class: GroupBNodeDlg
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to retrieve (see GroupB Enums in globals.h).
Returns: CString typed value.

Class: GroupBNodeDlg
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have changed since the last edit. This is useful for updating portions of the interface that depend on this node's data.
1.0 C++ Classes for Wasp Builder by Source File

Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since the last edit.

Class: GroupBNodeDlg
Function: UpdateTable(BOOL isInitializing=TRUE) : void
Description: Build m_waterTableCtrl and update data list. If isInitializing, then also initialize the backup data.
Input Args: isInitializing: if TRUE, then initialize m_backDataArray.

Class: GroupBNodeDlg
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: ar: the archive object to read/write to.

Class: GroupBNodeDlg
Member: m_surf_ntex : int
Description: Number of surface water exchange time functions. Needed so that the node knows how many entries to allocate for data. Set in global group B dialog.

Class: GroupBNodeDlg
Member: m_pore_ntex : int
Description: Number of pore water exchange time functions. Needed so that the node knows how many entries to allocate for data. Set in global group B dialog.

Class: GroupBNodeDlg
Member: m_dataArray : StringArray
Description: Data is organized as follows: m_dataArray: indexed 0..m_ntex. each elements is a StringArray with elements:\n 0. Area\n 1. Characteristic length.

Class: GroupBNodeDlg
Member: m_backDataArray : StringArray
Description: Backup versions for reverting changes when cancel is hit and when determining if the list has been changed (needed for building the changedParameter list.

Class: GroupBNodeDlg
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be sent back to the container.

11. Source File: GroupCDlg.h

Class: GroupCDlg
Function: GroupCDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group C card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.
12. Source File: groupcnodedlg.h

Class: GroupCDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card C data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupCDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward
compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: GroupCDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupCDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

12. Source File: groupcnodedlg.h

Class: GroupCNodeDlg
Function: GroupCNodeDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group C segment data dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: GroupCNodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to assign to (see GroupC Enums in globals.h).
Input Args: paramStr: CString typed value to assign.

Class: GroupCNodeDlg
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to retrieve (see GroupC Enums in globals.h).
Returns: CString typed value.
1.0 C++ Classes for Wasp Builder by Source File

Class: GroupCNodeDlg
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have changed since the last edit. This is useful for updating portions of the interface that depend on this node's data.
Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since the last edit.

Class: GroupCNodeDlg
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: ar: the archive object to read/write to.

Class: GroupCNodeDlg
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be sent back to the container.

13. Source File: GroupDDlg.h

Class: GroupDDlg
Function: GroupDDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global groupD card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: GroupDDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading cardD data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupDDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: GroupDDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.
Class: **GroupDDlg**
Function: NewTimefunc(int flowIdx, int timeIdx, CString newStr = "0 0") : void
Description: Add new time function given by newStr for flowIdx's flow array at position timeIdx.
Input Args: 
- **flowIdx**: The flow type to add to (see NFIELD in cardD).
- **timeIdx**: Where to add the new timefunction.
- **newStr**: The new time function
Returns: TRUE if successful.

Class: **GroupDDlg**
Function: UpdateSymType() : void
Description: When the simulation type is changed in card A, this will allocate or deallocate memory for the new number of systems.

Class: **GroupDDlg**
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: 
- **linelist**: output variable that will contain the results.

Class: **GroupDDlg**
Member: m_flowArrayLists : StringArray[6]
Description: One for each potential flow. If unused, SCALQ and CONVQ should be (0, 0) or empty. Each component is a list of lists with elements (SCALQ, CONVQ, [QT TQ]). Each item in the top-level list is a flow time function whose count is NINQ.

Class: **GroupDDlg**
Member: m_flowArrayLists : StringArray[6]
Description: Equivalent backup list. This will be set in OnInitDialog, and if the dialog is canceled, then m_flowArrayLists will be recovered from this.

Class: **GroupDDlg**
Member: m_currentSystem : int
Description: Use this rather than m_systemTableCtrl.GetActiveRow() to get the system being edited because OnEndlabeledEditSegList() doesn't seem to get called before OnClickSystemList is processed (it is probably a bug in my tablectrl, but this is a quick workaround).

Class: **GroupDDlg**
Member: m_systemBypassArray List StringArray
Description: Internal list of system bypass options

---

14. Source File: *groupnodedlg.h*

Class: **GroupDDlg**
Function: GroupDDnodeDlg(CWnd* pParent = NULL) : Constructor
Description: Displays group B exchange data dialog.
Input Args: **pParent**: the optional parent window.
Class: **GroupDNodeDlg**
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: *key*: the parameter to assign to (see GroupDEnums in globals.h).
Input Args: *paramStr*: CString typed value to assign.

Class: **GroupDNodeDlg**
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: *key*: the parameter to retrieve (see GroupDEnums in globals.h).
Returns: CString typed value.

Class: **GroupDNodeDlg**
Function: HasData() : BOOL
Description: Check if the exchange has flow data entered.
Returns: TRUE if there is flow data for this exchange.

Class: **GroupDNodeDlg**
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have changed since the last edit. This is useful for updating portions of the interface that depend on this node's data.
Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since the last edit.

Class: **GroupDNodeDlg**
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: *ar*: the archive object to read/write to.

Class: **GroupDNodeDlg**
Member: m_flowFieldArray : StringArray[6]
Description: The flows for a given field (6 fields, n flows, where n == ninq, the number of time functions for field i).

Class: **GroupDNodeDlg**
Member: m_nmaxtimefuncs : int
Description: max NINQ, the largest number of time functions used by particular field. This is needed to determine the number of columns needed for the table.

Class: **GroupDNodeDlg**
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be sent back to the container.
15. Source File: GroupEDlg.h

Class: GroupEDlg
Function: GroupEDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group E card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: GroupEDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card E data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupEDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: GroupEDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupEDlg
Function: UpdateSymType() : void
Description: When the simulation type is changed in card A, this will allocate or deallocate memory for the new number of systems.

Class: GroupEDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: GroupEDlg
Member: m_boundArrayList : StringArray[6]
Description: The data structure for each of the NSYS boundary concentration groups. 
IMPORTANT: I have hard-coded the max number of systems to 8; this will need to change if max NSYS can change.

Class: GroupEDlg
Member: m_nbbreaksArrayboundArrayList : CArray<int, int>
Description: Holds the count of breaks for each system of Group E data. This is populated by querying the dialog for all the segments' data.
16. Source File: groupenodedlg.h

Class: GroupENodeDlg
Function: GroupENodeDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group E segment data dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: GroupENodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to assign to (see GroupEEnums in globals.h).
Input Args: paramStr: CSTR typed value to assign.

Class: GroupENodeDlg
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to retrieve (see GroupEEnums in globals.h).
Returns: CSTR typed value.

Class: GroupENodeDlg
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have changed since the last edit. This is useful for updating portions of the interface that depend on this node's data.
Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since the last edit.

Class: GroupENodeDlg
Function: UpdateSymType() : void
Description: Called when the simulation type has changed.

Class: GroupENodeDlg
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: ar: the archive object to read/write to.

Class: GroupENodeDlg
Member: m_boundaryListArray : StringArray[8]
Description: The data structure for each of the NSYS boundary concentration groups. Each item is a StringArray list of boundary concentrations, further composed of a StringArray of time functions (val, time). The number of records for a given system's boundary concentrations yields the NOBC value. IMPORTANT: I have hard-coded the max number of systems to 8; this will need to change if max NSYS can change.

Class: GroupENodeDlg
Member: m_boundaryListArray : StringArray[8]
Description: If cancelled, backup from this array.
Class: GroupEENodeDlg
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be sent back to the container.

Class: GroupFDlg
Function: GroupFDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group F card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: GroupFDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card F data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupFDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: GroupFDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupFDlg
Function: GetSegIdxFromID(int sysIdx=-1, int segIdx=-1) : int
Description: Gets the corresponding index into the system-segment string array from a segmentID. Use the selections from the system and segment list boxes to make the determination if not explicitly given.
Input Args: sysIdx: The system to query (defaults to selected system in the system table).
Input Args: segIdx: The segment of interest (defaults to the selected segment in the segment table) (0 based).
Returns: The index into the system-segment string array m_systemListArray.

Class: GroupFDlg
Function: InitSystemListItem(int sysIdx=-1) : void
Description: Initializes the array for the selected system in the system list box.
Input Args: sysIdx: the system of interest.
1.0  C++ Classes for Wasp Builder by Source File

**Class**: GroupFDlg  
**Function**: InitSegmentArray(int sysIdx=-1, int segIdx=-1) : int  
**Description**: Initializes segment array list for the selected system (both system and segment choice are taken from the listboxes by default).  
**Input Args**:  
- **sysIdx**: the system of interest.  
- **segIdx**: the segment of interest (0 based).

**Class**: GroupFDlg  
**Function**: SetSegData(int sysnum, int segID, StringArray& newLoadFuncs) : void  
**Description**: Assigns to m_systemListArray the new load function for the given system and segment.  
**Input Args**:  
- **sysnum**: the system of interest.  
- **segID**: the segment of interest (0 based).  
- **newLoadFuncs**: the new load time function.

**Class**: GroupFDlg  
**Function**: UpdateSymType() : void  
**Description**: When the simulation type is changed in card A, this will allocate or deallocate memory for the new number of systems.

**Class**: GroupFDlg  
**Function**: CheckGlobalDataCompleteness(CStringList &linelist) : void  
**Description**: Return a list of possible holes in global data.  
**Input Args**:  
- **linelist**: output variable that will contain the results.

**Class**: GroupFDlg  
**Member**: m_systemListArray : StringArray[8]  
**Description**: Segment load data will be stored as a 2D array of segments and lists; each segment list item will have its point source loadings. Further, there will be one of these string arrays for each system. The first element of each array will be the ID of the segment that the array corresponds to. Do not use theID of the segment to index the array directly; rather the selected segmentID text should be tested against the first element of each array to see if that array string is the one to use. Important: NSYS is assumed to max out at 8; change this if later versions increase that value.

**Class**: GroupFDlg  
**Member**: m_backSystemListArray : StringArray[8]  
**Description**: Backup list--revert from this list if cancelled.

**Class**: GroupFDlg  
**Member**: m_currentSystem : int  
**Description**: Use this rather than m_systemTableCtrl.GetActiveRow() to get the system being edited because OnEndlabeleditSegList() doesn't seem to get called before OnClickSystemList is processed (it is probably a bug in my tablectrl, but this is a quick workaround).
18. Source File: GroupFNodeDlg.h

Class: GroupFNodeDlg
Function: GroupFNodeDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group F segment data dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: GroupFNodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to assign to (see GroupFEnums in globals.h).
Input Args: paramStr: CStr typed value to assign.

Class: GroupFNodeDlg
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to retrieve (see GroupFEnums in globals.h).
Returns: CString typed value.

Class: GroupFNodeDlg
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have
changed since the last edit. This is useful for updating portions of the interface that
depend on this node's data.
Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since
the last edit.

Class: GroupFNodeDlg
Function: UpdateSymType() : void
Description: Called when the simulation type has changed.

Class: GroupFNodeDlg
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: ar: the archive object to read/write to.

Class: GroupFNodeDlg
Member: m_systemListArray : StringArray[8]
Description: Internal storage of group F data. IMPORTANT: I have hard-coded the max number of
systems to 8; this will need to change if max NSYS can change.

Class: GroupFNodeDlg
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be
sent back to the container.
19. Source File: GroupGDlg.h

Class: GroupGDlg
Function: GroupGDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group G card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: GroupGDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card G data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupGDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: GroupGDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupGDlg
Function: RedrawTable() : void
Description: Updates m_paramTableCtrl.

Class: GroupGDlg
Function: UpdateHelpLabel() : void
Description: Updates help label text based on the selected parameter.

Class: GroupGDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: GroupGDlg
Function: SetNParams() : void
Description: Assigns to m_nparams the number of parameters for this system/
Class: GroupGDlg
Member: m_nparams : int
Description: The number of model parameters being considered. Eutro = 14, Toxi = 18, Metals = 32 (1-33 w/out 30). Read from WaspClass or whenInitDialog detects a model simulation type change will change this.

Class: GroupGDlg
Member: m_paramValueList : double[32]
Description: Storage of parameter values. Hard-code upper-limit to 32; be sure to change this if the number of possible parameters for a given simulation type increases.

Class: GroupGDlg
Member: m_pnameValueList : double[32]
Description: Storage of parameter names. Hard-code upper-limit to 32; be sure to change this if the number of possible parameters for a given simulation type increases.

20. Source File: groupnode.h

Class: GroupGNodeDlg
Function: GroupGNodeDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group G segment data dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: GroupGNodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Assign a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to assign to (see GroupGEnums in globals.h).
Input Args: paramStr: CString typed value to assign.

Class: GroupGNodeDlg
Function: GetParameter(int key) : void
Description: Retrieve a parameter a value. Use parameter enums found in Globals.h.
Input Args: key: the parameter to retrieve (see GroupGEnums in globals.h).
Returns: CString typed value.

Class: GroupGNodeDlg
Function: GetChangedParametersArray() : StringArray
Description: Retrieve a flat list of (parameter, value) pairs containing all parameters that have changed since the last edit. This is useful for updating portions of the interface that depend on this node's data.
Returns: A flat list of (parameter, value) pairs containing all parameters that have changed since the last edit.

Class: GroupGNodeDlg
Function: UpdateSymType() : void
Description: Called when the simulation type has changed.
1.0 C++ Classes for Wasp Builder by Source File

Class: GroupGNodeDlg
Function: UpdateTable(Serialize(CArchive &ar) : void
Description: Save or retrieve state from archive object.
Input Args: ar: the archive object to read/write to.

Class: GroupGNodeDlg
Member: m_nParams : int
Description: The number of parameters in the simulation. Determined by the sim type (eutro, toxi, metals). Assigned during initialization.

Class: GroupGNodeDlg
Member: m_descripArray : CString[32]
Description: All the group G parameter descriptions. IMPORTANT: I have hard-coded the max number of systems to 32 (the number of params in meta); this will need to change if max number of parameters ever changes.

Class: GroupGNodeDlg
Member: m_paramArray : double[32]
Description: All the group G parameter values. IMPORTANT: I have hard-coded the max number of systems to 32 (the number of params in meta); this will need to change if max number of parameters ever changes.

Class: GroupGNodeDlg
Member: m_changedParameters : StringArray
Description: Keeps track of changes in the dialog so that after "ok", only the new stuff needs to be sent back to the container.

21. Source File: GroupHDlg.h

Class: GroupHDlg
Function: GroupHDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group H card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: GroupHDlg
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card H data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: GroupHDlg
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.
Class: GroupHDlg
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.

Class: GroupHDlg
Function: AllocateSystemMemory(int sysIdx=-1) : void
Description: Allocates space in m_constantListArray for a new system (initially no systems have space allocated).
Input Args: sysIdx: defaults to the m_currentSystem. The system to allocate space for.

Class: GroupHDlg
Function: AllocateDatafieldMemory(int sysIdx=-1, int fieldIdx=-1) : void
Description: Allocates space in m_constantListArray for a new field in a system (initially no fields have space allocated).
Input Args: sysIdx: defaults to the m_currentSystem. The system to allocate space for.
Input Args: fieldIdx: defaults to the m_currentField. The field to allocate space for.

Class: GroupHDlg
Function: UpdateSymType() : void
Description: When the simulation type is changed in card A, this will allocate or deallocate memory for the new number of systems.

Class: GroupHDlg
Function: UpdateFieldTable() : void
Description: Updates m_datafieldsTableCtrl, the field list.

Class: GroupHDlg
Function: UpdateConstTable() : void
Description: Updates m_constValTableCtrl, the table of constants for a given system and field.

Class: GroupHDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: GroupHDlg
Function: m_constantListArray : StringArray[9]
Description: Internal storage. One string array for each system, plus an extra for global constants.
Each element of the array is:
1. CHNAME [system name] 2. StringArray of field names, repeated NFLD times.
3. FLDNAME 4. StringArray of constant data, repeated NCONS times.
5. TNAME [name of constant] ii. ISC number (needed because constants may be sparsely populated)
iii. CONST

Class: GroupHDlg
Member: m_backupListArray : StringArray[9]
Description: If cancelled, revert from this list.
1.0 C++ Classes for Wasp Builder by Source File

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupHDlг</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member</td>
<td>m_currentSystem : int</td>
</tr>
<tr>
<td></td>
<td>Current system being edited.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupHDlг</th>
</tr>
</thead>
<tbody>
<tr>
<td>Member</td>
<td>m_currentField : int</td>
</tr>
<tr>
<td></td>
<td>Current field being edited.</td>
</tr>
</tbody>
</table>

### 22. Source File: `GroupIDlg.h`

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>GroupIDlg(ProjectEditDlg <em>prjDlg, CWnd</em> pParent = NULL) : Constructor</td>
</tr>
<tr>
<td>Description</td>
<td>Interface for the global groupID card (ie, displays card data that doesn't include most individual segment or node data.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>prjDlg</code>: backpointer so info from other group dialogs can be accessed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>LoadFromWC(WaspClass *wc) : BOOL</td>
</tr>
<tr>
<td>Description</td>
<td>Initialize by reading cardID data from WaspClass.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>wc</code>: WaspClass created by importing a WASP input data file.</td>
</tr>
<tr>
<td>Returns</td>
<td>TRUE if successful.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>ReadAR(CArchive&amp; ar, int version) : BOOL</td>
</tr>
<tr>
<td>Description</td>
<td>Update state from archive data.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>ar</code>: the class data.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>version</code>: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.</td>
</tr>
<tr>
<td>Returns</td>
<td>TRUE if successful.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>WriteAR(CArchive&amp; ar) : BOOL</td>
</tr>
<tr>
<td>Description</td>
<td>Write state to archive data. Always write using the latest version's format.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>ar</code>: the class data.</td>
</tr>
<tr>
<td>Returns</td>
<td>TRUE if successful.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>UpdateHelpLabel() : void</td>
</tr>
<tr>
<td>Description</td>
<td>When a time function is selected, this will update the help text label.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class</th>
<th>GroupIDlg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>SetNumTimefuncs(int which) : char**</td>
</tr>
<tr>
<td>Description</td>
<td>Set the member m_nftimefuncs and return an array containing either the name,ISC, or descrip.</td>
</tr>
<tr>
<td>Input Args</td>
<td><code>which</code>: 0 == PNAME, 1 ==ISC, 2 == descrip.</td>
</tr>
</tbody>
</table>
Class: **GroupIDlg**
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: **GroupIDlg**
Member: m_ntimefuncs : int
Description: The number of model parameters being considered. Eutro = 14, Toxi = 18, Metals = 32 (1-33 w/out 30). Read from WaspClass or whenInitDialog detects a model simulation type change will change this.

Class: **GroupIDlg**
Member: m_timefuncValueList : CString[32]
Description: Storage of parameter values. Hard-code upper-limit to 32; be sure to change this if the number of possible parameters for a given simulation type increases.

Class: **GroupIDlg**
Member: m_anameValueList : CString[32]
Description: Storage of parameter names. Hard-code upper-limit to 32; be sure to change this if the number of possible parameters for a given simulation type increases.

23. Source File: **GroupIDlg.h**

Class: **GroupIDlg**
Function: GroupIDlg(ProjectEditDlg *prjDlg, CWnd* pParent = NULL) : Constructor
Description: Interface for the global group J card (ie, displays card data that doesn't include most individual segment or node data.
Input Args: prjDlg: backpointer so info from other group dialogs can be accessed.

Class: **GroupIDlg**
Function: LoadFromWC(WaspClass *wc) : BOOL
Description: Initialize by reading card J data from WaspClass.
Input Args: wc: WaspClass created by importing a WASP input data file.
Returns: TRUE if successful.

Class: **GroupIDlg**
Function: ReadAR(CArchive& ar, int version) : BOOL
Description: Update state from archive data.
Input Args: ar: the class data.
Input Args: version: Read the archive differently based on version number (allows for backward compatibility with older files). Note the latest version format will always be written out.
Returns: TRUE if successful.

Class: **GroupIDlg**
Function: WriteAR(CArchive& ar) : BOOL
Description: Write state to archive data. Always write using the latest version's format.
Input Args: ar: the class data.
Returns: TRUE if successful.
1.0 C++ Classes for Wasp Builder by Source File

Class: GroupJDlg
Function: AddSystem(int newSysIdx) : void
Description: Create space for a new system in m_ICListArray by deleting previous contents.
Input Args: newSysIdx: the index of the new system.

Class: GroupJDlg
Function: AddSegment() : void
Description: Create space for a new segment. Usually called when the user has added a new segment to the network. The new segment goes at the end of the list (can't add in the middle).

Class: GroupJDlg
Function: AddSegment(int whichID) : void
Description: Create space for a new segment. Usually called when the user has deleted a segment from the network.
Input Args: whichID: the 0-based segmentID.

Class: GroupJDlg
Function: UpdateSymType() : void
Description: Initialize system names.

Class: GroupJDlg
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: output variable that will contain the results.

Class: GroupJDlg
Member: m_ICListArray : StringArray[6]
Description: The data structure for each of the system initial condition components. Contents by index:
0: CHEML
1: IFIELD
2: DSED
3: CMAX
4: TITLE
5: StringArray of Segments:
0: A StringArray of IC data: ANAME
ii: C
iii: DISSF
1: <etc>

IMPORTANT: I have hard-coded the max number of systems to 8; this will need to change if max NSYS can change.

Class: GroupJDlg
Member: m_backupICListArray : StringArray[6]
Description: Backup lists if canceled.
Member: m_currentSystem : int
Description: Use this rather than m_systemTableCtrl.GetActiveRow() to get the system being edited because OnEndlabeleditSegList() doesn't seem to get called before OnClickSystemList is processed (it is probably a bug in my tablectrl, but this is a quick workaround)

24. Source File: InputPlotBDlg.h

Class: InputPlotBDlg
Function: InputPlotBDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group B plot dialog.
Input Args: doc: The network document so all card data can be accessed.
Class: InputPlotBDlg
Member: m_waterArrayList : StringArray*
Description: This is a pointer to data in the group B dialog.

Class: InputPlotBDlg
Member: m_exchList : CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>*
Description: The list of all valid exchanges in the network (ie, those that have data).

25. Source File: InputPlotCDlg.h

Class: InputPlotCDlg
Function: InputPlotCDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group C plot dialog.
Input Args: doc: The network document so all card data can be accessed.

26. Source File: InputPlotDDlg.h

Class: InputPlotDDlg
Function: InputPlotDDDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays groupD plot dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: InputPlotDDlg
Function: GetWaterFlowLabel(int whichField) : CString
Description: Get the water flow description (Advective Flow, etc).
Input Args: whichField: the water flow field of interest.

Class: InputPlotDDlg
Member: m_exchList : CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>*
Description: The list of all valid exchanges in the network (ie, those that have data).

27. Source File: InputPlotEDlg.h

Class: InputPlotEDlg
Function: InputPlotEDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group F plot dialog.
Input Args: doc: The network document so all card data can be accessed.

28. Source File: InputPlotFDlg.h

Class: InputPlotFDlg
Function: InputPlotFDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group B plot dialog.
Input Args: doc: The network document so all card data can be accessed.
29. Source File: InputPlotIDlg.h

Class: InputPlotIDlg
Function: InputPlotIDlg(CWaspBuilderDoc *doc, CWnd* pParent = NULL) : Constructor
Description: Displays group I plot dialog.
Input Args: doc: The network document so all card data can be accessed.

Class: InputPlotIDlg
Function: SetNumTimefuncs(int which) : char**
Description: Set the member m_ntimefuncs and return an array containing either the name, ISC, or
descrip. See GroupIDlg.
Input Args: which: 0 == PNAME, 1 ==ISC, 2 == descrip

Class: InputPlotIDlg
Member: m_validTimefuncsList : CList<int, int>
Description: Get a list of all timefunctions for this project.

Class: InputPlotIDlg
Member: m_ntimefuncs : int
Description: The number of model parameters being considered. Eutro = 14, Toxi = 18, Metals =
32 (1-33 w/out 30).

30. Source File: labelnodedlg.h

Class: LabelNodeDlg
Function: LabelNodeDlg(CWnd* pParent = NULL) : Constructor
Description: A dialog for interacting with label nodes.

Class: LabelNodeDlg
Function: GetChangedParametersArray(CStringList &retList) : void
Description: Retrieves parameters related to exchange nodes that have been changed, allowing the
document to update data dependent on this node.
Input Args: retList: A list of key, value pairs.

Class: LabelNodeDlg
Function: GetFont() : CFont&
Description: Retrieves the font object associated with this node.
Returns: The font for this node.

Class: LabelNodeDlg
Function: SetFont(CFont&) : void
Description: Assign a new font for this label node.
Input Args: The font object to assign to m_labelFont.
31. Source File: MultiTrack.h

Class: LabelNodeDlg
Function: SetParameter(int key, LPCWSTR paramStr) : void
Description: Parameter 'key' is assigned value 'value'.
Input Args: key: the parameter to assign.
Input Args: value: the parameter's new value.

Class: LabelNodeDlg
Function: GetParameter(int key) : CString
Description: Retrieve value of parameter 'key'.
Input Args: key: the parameter to query.
Returns: value: the parameter's value.

Class: LabelNodeDlg
Function: Serialize(CArchive& ar) : void
Description: Stores/Retrieves the label's data from file object.
Input Args: ar: the file object.

Class: LabelNodeDlg
Member: m_changedParameters : StringArray
Description: On OK will update the string of changed parameters.

32. Source File: OutputDlg.h

Class: OutputDlg
Function: OutputDlg(CWaspBuilderDoc* pDoc, CString basePathName, CWnd* pParent = NULL) : Constructor
Description: Stores information related to exchange nodes. See CntrItem for additional information.
Input Args: pDoc: the document with all WASP data.
Input Args: basePathName: Name of the WASP output file.

Class: OutputDlg
Function: InitializeTypeCombo() : void
Description: Initialize output type combo box based on output files present.

Class: OutputDlg
Function: GetOutputTypeAndExtName(CString typeStr, CString *fname) : OutputTypeEnum
Description: Return the output type based on the file name.
Returns: The output type based on the file name.
1.0 C++ Classes for Wasp Builder by Source File

Class: OutputDlg
Function: InitializeSegAndVarList() : void
Description: Populate m_variableList and m_segmentList.

Class: OutputDlg
Function: UpdateOutputView(ViewModeEnum doPrint=PRINT) : void
Description: Populate m_outputView with results.

Class: OutputDlg
Function: CheckDiffSensButton() : void
Description: If there are multiple runs, then this will sensitize the check difference button.

Class: OutputDlg
Function: ReadFiles() : void
Description: Load the output data for the selected group of files into m_waspOutputData.

33. Source File: PaletteDlg.h

Class: PaletteDlg
Function: PaletteDlg(CWaspBuilderDoc *pDoc, CWnd* pParent = NULL) : Constructor
Description: Displays the palette dialog with an array of icons that can be place on the network view.
Alerts the document that the user wants to add a new node.
Input Args: pDoc: the document with all WASP data.

34. Source File: ProjectEditDlg.h

Class: ProjectEditDlg
Function: PaletteDlg(CWaspBuilderDoc *pDoc, CWnd* pParent = NULL) : Constructor
Description: Displays the global card dialog. The dialog will coordinate changes to segments and exchanges if needed.
Input Args: pDoc: the document with all WASP data.

Class: ProjectEditDlg
Function: SetButtonSensitivity() : void
Description: If the simulation type has not been set in card A, then this routine will desensitize groups B-J.

35. Source File: RangeDlg.h

Class: RangeDlg
Function: RangeDlg(CString fileName = "", CWnd* pParent = NULL) : Constructor
Description: Displays a dialog that allows the user to set ranges for potentially any card group.
NOTE: this dialog is currently only implemented for group J, pending a decision on whether it makes sense to do the same for the other cards.
Input Args: pContainer: the text file containing range data.
Class: RangeDlg
Function: SetFileName(CString fileName) : void
Description: Set the source file name for range values.

Class: RangeDlg
Function: GetFileName() : CString
Description: Get the source file name for range values.

Class: RangeDlg
Function: UpdateTableData() : void
Description: Populate each of the card tables (m_jTableCtrl).
Returns: m_fileName.

Class: RangeDlg
Function: SaveTableData() : void
Description: Write range files to the current file name.

36. Source File: RCSClass.h

Class: RCSClass
Function: RCSClass(CString fileName) : Constructor
Description: ReadRCS information for file 'fileName'. Populates m_description, m_versionList, m_dateList, and m_logMessageList.
Input Args: fileName: The name of the file to query.

Class: RCSClass
Function: ~RCSClass() :Destructor
Description: Because the constructor includes a chdir to the directory of the file, the destructor will return the program to the current dir.

Class: RCSClass
Function: InitData() : void
Description: Populates m_description, m_versionList, m_dateList, and m_logMessageList.

Class: RCSClass
Function: COVersion(float ver) : BOOL
Description: Force a check out of the specified version of the file.
Returns: TRUE if successful.

Class: RCSClass
Function: CIVersion(float ver, CString symname, CString log) : BOOL
Description: Perform a check in of the file with the given version number, symbolic name, and log message.
Returns: TRUE if successful.

Class: RCSClass
Function: DeleteVersion(float ver) : BOOL
Description: Delete the given version of the file fromRCS. Reinitialize data.
Returns: TRUE if successful.
1.0 C++ Classes for Wasp Builder by Source File

Class: RCSClass
  Member: m_description : CString
  Description: The RCS description for m_fileName.

Class: RCSClass
  Member: m_versionList : CList<float, float&>
  Description: All the RCS versions numbers of m_fileName.

Class: RCSClass
  Member: m_symNameList : CList<CString, CString&>
  Description: All the RCS symbolic names of m_fileName.

Class: RCSClass
  Member: m_dateList : CList<CString, CString&>
  Description: All the RCS creation dates of each version of m_fileName.

Class: RCSClass
  Member: m_logMessageList : CList<CString, CString&>
  Description: All the RCS log messages of m_fileName.

37. Source File: RCSOpenDlg.h

Class: RCSOpenDlg
  Function: RCSOpenDlg(CString fileName, CWnd* pParent = NULL) : Constructor
  Description: Displays a dialog box for opening up versions of 'fileName'. If Ok'ed, the dialog will open the version, destroying the original contents of the file.
  Input Args: fileName: the name of the file to retrieve RCS versions of.

Class: RCSOpenDlg
  Function: InitGui() : void
  Description: Populates the dialog box with log messages, versions, and file description.

38. Source File: RCSSaveDlg.h

Class: RCSSaveDlg
  Function: RCSSaveDlg(CString fileName, CWnd* pParent = NULL) : Constructor
  Description: Displays a dialog box for saving a new versions of 'fileName'. If Ok'ed, the dialog will check in the new version.
  Input Args: fileName: the name of the file to create a new version of.

Class: RCSSaveDlg
  Function: InitGui() : void
  Description: Populates the dialog box with log messages, versions, and file description.
39. Source File: ReportData.h

**Class:** NodeData
**Function:** NodeData(int id=0, CString name="", int nParams=0) : Constructor
**Description:** Stores information about a WASP segment. Currently this is just the alias and a currently unused list of threshold values for a given parameter. *NOTE:* Do not confuse with the output data class. This just holds info about threshold data.

**Input Args:**
- *id:* The segmentID (Assumes first node starts at 1).
- *name:* The segment's alias.
- *nParams:* If individual segment thresholds are to be used, then assign this to be the total number of parameters of all the output files.

**Class:** NodeData
**Function:** NodeData(NodeData&) : Copy Constructor

**Class:** NodeData
**Function:** operator=(NodeData&) : NodeData&

**Class:** NodeData
**Function:** operator<<(ostream &ostr, NodeData &nd) : ofstream&

**Class:** NodeData
**Function:** operator>>(ifstream &istr, NodeData &nd) : ifstream&

**Class:** NodeData
**Member:** m_parameterArray : CArray<double, double>
**Description:** Each parameter has a threshold value for a given parameter at this segment. Ordering should be the same as in ParameterData's m_parameterNamesArray.

**Class:** ParameterData
**Function:** ParameterData() : Constructor
**Description:** Manages the list of nodes (segments).

**Class:** NodeData
**Function:** operator<<(ostream &ostr, ParameterData &pd) : ofstream&

**Class:** NodeData
**Function:** operator>>(ifstream &istr, ParameterData &pd) : ifstream&

**Class:** ParameterData
**Function:** AddNode(int segID, CString segName="") : BOOL
**Description:** Add a new segment (node) to the list with alias 'segName'.
**Returns:** TRUE if successful.

**Class:** ParameterData
**Function:** DeleteNode(int segID) : BOOL
**Description:** Delete a segment (node) from the list with segmentID 'segID'.
**Returns:** TRUE if successful.
1.0 C++ Classes for Wasp Builder by Source File

Class: ParameterData
Member: m_parameterNamesArray : CArray<CString, LPCSTR>
Description: The names of all the output parameters.

Class: ParameterData
Member: m_nodeData : CMap<int, int, NodeData, NodeData&>
Description: List of all segments. Since the node data can be sparse, use a dictionary for storage.

Class: NodeOutputData
Function: NodeOutputData() : Constructor
Description: Output data for a given period of time.

Class: NodeOutputData
Function: NodeOutputData(NodeOutputData&) : Copy Constructor

Class: NodeOutputData
Function: operator=(NodeOutputData&) : NodeOutputData&

Class: NodeOutputData
Function: operator[](int index) : int
Description: Shortcut for dereferencing m_periodOutputData.
Input Args: index: dereference the 'index' element of m_periodOutputData.
Returns: m_periodOutputData[index]

Class: NodeOutputData
Member: m_periodOutputData : CArray<double, double>
Description: Array of time period output for a parameter and station.

Class: VariableOutputData
Function: VariableOutputData() : Constructor
Description: Output data for a parameter.

Class: VariableOutputData
Function: VariableOutputData(VariableOutputData&) : Copy Constructor

Class: VariableOutputData
Function: operator=(VariableOutputData&) : VariableOutputData&

Class: NodeOutputData
Function: operator[](int index) : NodeOutputData&
Description: Shortcut for dereferencing m_nodeOutputData.
Input Args: index: dereference the 'index' element of m_periodOutputData.
Returns: m_periodOutputData[index]

Class: NodeOutputData
Function: GetMin(int whichSeg=-1) : double
Input Args: whichSeg: The segment to get the minimum of segment data from. Defaults to all segments. Useful for setting graph axis scale bounds.
Returns: The minimum of the segment's data.
Class: NodeOutputData
Function: GetMax(int whichSeg=-1) : double
Input Args: whichSeg: The segment to get the maximum of segment data from. Defaults to all segments. Useful for setting graph axis scale bounds.
Returns: The maximum of the segment’s data.

Class: NodeOutputData
Member: m_varName : CString
Description: The name of the parameter this object represents.

Class: NodeOutputData
Member: m_varUnits : CString
Description: The name of the units of the parameter this object represents.

Class: NodeOutputData
Member: m_nodeOutputData : CArray<NodeOutputData, NodeOutputData&>
Description: The segment data for this parameter.

Class: WaspOutputData
Function: WaspOutputData(CString pathname="") : Constructor
Description: Manages a complete WASP output data set.
Input Args: pathname: the name of the dataset. If not given, be sure to call SetPathName before running ReadInputData.

Class: WaspOutputData
Function: WaspOutputData(WaspOutputData&) : Copy Constructor

Class: WaspOutputData
Function: operator=(WaspOutputData&) : WaspOutputData&

Class: WaspOutputData
Function: operator<<(ofstream &ostr, WaspOutputData &wod) : ostream&

Class: WaspOutputData
Function: operator>>(ifstream &istr, WaspOutputData &wod) : ifstream&

Class: WaspOutputData
Function: SetPathName(CString pathname) : void
Description: Gets the base name of the WASP files which will be used to read input and output data.
Input Args: pathname: Name of the WASP output or input file (doesn't matter because the function will separate out the basename, which is what is used to read the transient and simulation result files.

Class: WaspOutputData
Function: ReadInputData() : BOOL
Description: Parses the input file for run type and header information.
Returns: TRUE if successful.
1.0 C++ Classes for Wasp Builder by Source File

Class: WaspOutputData
Function: ReadOutputData() : BOOL
Description: Tests for existence of various output files and calls ReadOutputData(FILE *fp, 
CArray<VariableOutputData, VariableOutputData&> *varDataArray, 
CArray<double, double> *periodArray) for each type.
Returns: TRUE if successful.

Class: WaspOutputData
Function: ReadOutputData(FILE *fp, CArray<VariableOutputData, VariableOutputData&> 
*varDataArray, CArry<double, double> *periodArray) : BOOL
Description: Loads output from file ‘fp’ into either m_variableOutputTRNData or 
m_variableOutputSIMData.
Input Args: fp: The file to read output from.
Input Args: varDataArray: Pointer to either m_variableOutputTRNData or 
m_variableOutputSIMData, depending on the type of output being read.
Input Args: periodArray: Pointer to either m_periodTRNArrary or m_periodSIMArray, depending 
on the type of output being read.
Returns: TRUE if successful.

Class: WaspOutputData
Function: GetVal(BOOL useTRN, int whichVar, int whichSeg, int whichPeriod, double *retval) 
: BOOL
Description: Use this procedure to look up data. This will make sure that there are no out of bounds 
array conditions. Return TRUE if lookup succeeded.
Input Args: useTRN: If TRUE, then use m_variableOutputTRNData; otherwise use 
m_variableOutputSIMData.
Input Args: whichVar: The index of the variable of interest.
Input Args: whichSeg: The index of the segment of interest.
Input Args: whichPeriod: The index of the period of interest.
Input Args: retval: The result of the look up.
Returns: TRUE if successful.

Class: WaspOutputData
Function: GetMin(BOOL useTRN, int whichVar, int whichSeg, double *retval) : BOOL
Description: Use this procedure to look up the minimum value of this parameter and segment over 
all available time periods. This will make sure that there are no out of bounds array 
conditions. Return TRUE if lookup succeeded.
Input Args: useTRN: If TRUE, then use m_variableOutputTRNData; otherwise use 
m_variableOutputSIMData.
Input Args: whichVar: The index of the variable of interest.
Input Args: whichSeg: The index of the segment of interest. Use -1 for all segments.
Input Args: retval: The result of the look up.
Returns: TRUE if successful.
Class: WaspOutputData
Function: GetMax(BOOL useTRN, int whichVar, int whichSeg, double *retval) : BOOL
Description: Use this procedure to look up the maximum value of this parameter and segment over all available time periods. This will make sure that there are no out of bounds array conditions. Return TRUE if lookup succeeded.
Input Args: useTRN: If TRUE, then use m_variableOutputTRNData; otherwise use m_variableOutputSIMData.
Input Args: whichVar: The index of the variable of interest.
Input Args: whichSeg: The index of the segment of interest. Use -1 for all segments.
Input Args: retval: The result of the look up.
Returns: TRUE if successful.

Class: WaspOutputData
Function: GetName(BOOL useTRN, int whichVar) : CString
Description: Get the name of the parameter indexed by 'whichVar'.
Input Args: useTRN: If TRUE, then use m_variableOutputTRNData; otherwise use m_variableOutputSIMData.
Input Args: whichVar: The index of the variable of interest.
Returns: "<error>" if lookup failed; otherwise the name of the parameter.

Class: WaspOutputData
Function: GetUnits(BOOL useTRN, int whichVar) : CString
Description: Get the units of the parameter indexed by 'whichVar'.
Input Args: useTRN: If TRUE, then use m_variableOutputTRNData; otherwise use m_variableOutputSIMData.
Input Args: whichVar: The index of the variable of interest.
Returns: "<error>" if lookup failed; otherwise the units of the parameter.

Class: WaspOutputData
Member: m_variableOutputTRNData : CArray<VariableOutputData, VariableOutputData&>
Description: WASP output data storage. Dereference data using Var->Seg->Period.

Class: WaspOutputData
Member: m_variableOutputSIMData : CArray<VariableOutputData, VariableOutputData&>
Description: WASP output data storage. Dereference data using Var->Seg->Period.

Class: WaspOutputData
Member: m_periodTRNArray : CArray<double, double&>
Description: WASP output timestep intervals for transient output.

Class: WaspOutputData
Member: m_periodSIMArray : CArray<double, double&>
Description: WASP output timestep intervals for simulation output.

Class: InputError
Function: InputError(CString msg="") : Constructor
Description: Any parse errors should throw this object.
Input Args: msg: Error message to output.
1.0 C++ Classes for Wasp Builder by Source File

Class: InputError
Function: ErrorMsg() : CString
Description: Returns The error cause (m_errorMessage).
Returns: The error cause.

40. Source File: SegmentChooserDlg.h

Class: SegmentChooserDlg
Function: SegmentChooserDlg(CStringList *possibleSegments, CWnd* pParent = NULL : Constructor
Description: A dialog that allows the user to select multiple segments from a list.
Input Args: possibleSegments: List of segments to choose from.

Class: SegmentChooserDlg
Member: m_possibleSegments : CStringList*
Description: This is the list of segments to chose from.

Class: SegmentChooserDlg
Member: m_additionalSegments : CStringList
Description: This is the list of segments that the user selected. Populated if OK button pressed.

41. Source File: SegmentInsertDlg.h

Class: SegmentInsertDlg
Function: SegmentInsertDlg(CWnd* pParent = NULL, int oldID = -1, int newID = -1 : Constructor
Description: If a segmentID is changed, the user has the option of switching the segment's data with the segment data of the newID. Otherwise the user can move the segment to the new location and shuffle the other segments that are in between.
Input Args: oldID: The original segmentID.
Input Args: newID: The new segmentID.

42. Source File: SegmentNodeDlg.h

Class: SegmentNodeDlg
Function: SegmentNodeDlg(CWaspBuilderDoc* pContainer, CWnd* pParent = NULL) : Constructor
Description: Stores information related to segment nodes. See CntrItem for additional information.

Class: SegmentNodeDlg
Function: GetChangedParametersArray(CStringList &retList) : void
Description: Retrieves parameters related to segment nodes that have been changed, allowing the document to update data dependent on this node.
Input Args: retList: A list of key, value pairs.
Class: SegmentNodeDlg
Function: UpdateParameters(CMap<int, int, CString, CString>& &parametersMap) : void
Description: Set new parameters from the parameter map.
Input Args: parametersMap: A map of key, value pairs.

Class: SegmentNodeDlg
Function: SetParameter(int key, LPCTSTR paramStr) : void
Description: Parameter 'key' is assigned value 'value'.
Input Args: key: the parameter to assign.
Input Args: value: the parameter's new value.

Class: SegmentNodeDlg
Function: GetParameter(int key) : CString
Description: Retrieve value of parameter 'key'.
Input Args: key: the parameter to query.
Returns: value: the parameter's value.

Class: SegmentNodeDlg
Function: UpdateSymType() : void
Description: Alert segment that the simulation type has changed.

Class: SegmentNodeDlg
Function: Serialize(CArchive& ar) : void
Description: Stores/Retrieves the dialog's data from file object.
Input Args: ar: the file object.

Class: SegmentNodeDlg
Member: m_changedParameters : StringArray
Description: On OK will update the string of changed parameters.

43. Source File: SensAnalysisParamsDlg.h

Class: SensAnalysisParamsDlg
Function: SensAnalysisParamsDlg(CWnd* pParent = NULL) : Constructor
Description: Allows the user to set an optional title string for the sensitivity analysis runs. Also this is where the user indicates if a base run should be executed. After OK'ed, query m_useBasecase and m_title2 for results.

44. Source File: SensitivityDlgB.h

Class: SensitivityDlgB
Function: SensitivityDlgB(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group B options for sensitivity analysis runs.
1.0  C++ Classes for Wasp Builder by Source File

Class:    SensitivityDlgB
Function:  UpdateCurVal() : void
Description: Refresh the value of the current water field, exchange, and parameter.

Class:    SensitivityDlgB
Function:  UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.

Class:    SensitivityDlgB
Member:    m_waterArrayList : StringArray*
Description: This points to either the pore or surface water data.

Class:    SensitivityDlgB
Member:    m_exchList : CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>*
Description: The list of all valid exchanges in the network (ie, those that have data).

Class:    SensitivityDlgB
Member:    m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

45. Source File:    SensitivityDlgC.h

Class:    SensitivityDlgC
Function:  SensitivityDlgC(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group C options for sensitivity analysis runs.

Class:    SensitivityDlgC
Function:  UpdateCurVal() : void
Description: Refresh the value of the selected parameter of the selected segment.

Class:    SensitivityDlgC
Function:  UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.

Class:    SensitivityDlgC
Member:    m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

46. Source File:    SensitivityDlgD.h

Class:    SensitivityDlgD
Function:  SensitivityDlgD(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting groupD options for sensitivity analysis runs.
Class: SensitivityDlgD
Function: UpdateCurVal() : void
Description: Refresh the value of the selected flow and time function.

Class: SensitivityDlgD
Function: UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.

Class: SensitivityDlgD
Member: m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

47. Source File: SensitivityDlgE.h

Class: SensitivityDlgE
Function: SensitivityDlgE(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group E options for sensitivity analysis runs.

Class: SensitivityDlgE
Function: UpdateCurVal() : void
Description: Refresh the value of the selected segment and system.

Class: SensitivityDlgE
Function: UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.

Class: SensitivityDlgE
Member: m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

48. Source File: SensitivityDlgF.h

Class: SensitivityDlgF
Function: SensitivityDlgF(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group F options for sensitivity analysis runs.

Class: SensitivityDlgF
Function: UpdateCurVal() : void
Description: Refresh the value of the selected segment and system.

Class: SensitivityDlgF
Function: UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.
1.0 C++ Classes for Wasp Builder by Source File

Class: SensitivityDlgF
Member: m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

49. Source File: SensitivityDlgG.h

Class: SensitivityDlgG
Function: SensitivityDlgG(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group G options for sensitivity analysis runs.

Class: SensitivityDlgG
Function: LoadParameterCombo() : void
Description: Load the parameter combo box with all the parameter types for the model simulation type.

Class: SensitivityDlgG
Function: UpdateCurVal() : void
Description: Refresh the value of the selected segment and parameter.

Class: SensitivityDlgG
Function: UpdateValuesList() : void
Description: Build the list of intervals based on min, max, and number of intervals.

Class: SensitivityDlgG
Member: m_parameterList : CList<int, int&>
Description: Only count non-zero parameters. This list has the indices into the group G parameter arrays.

Class: SensitivityDlgG
Member: m_initialValue : double
Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

50. Source File: SensitivityDlgH.h

Class: SensitivityDlgH
Function: SensitivityDlgH(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor
Description: Dialog for selecting group H options for sensitivity analysis runs.

Class: SensitivityDlgH
Function: UpdateCurVal() : void
Description: Refresh the value of the selected segment and parameter.
Class: **SensitivityDlgH**

Function: **UpdateValuesList() : void**

Description: Build the list of intervals based on min, max, and number of intervals.

Class: **SensitivityDlgH**

Member: `m_initialValue : double`

Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

51. **Source File:** SensitivityDlgJ.h

Class: **SensitivityDlgJ**

Function: **SensitivityDlgJ(CWaspBuilderDoc* pDoc, CWnd* pParent = NULL) : Constructor**

Description: Dialog for selecting group J options for sensitivity analysis runs.


Class: **SensitivityDlgJ**

Function: **UpdateCurVal() : void**

Description: Refresh the value of the selected segment, parameter and system.

Class: **SensitivityDlgJ**

Function: **UpdateValuesList() : void**

Description: Build the list of intervals based on min, max, and number of intervals.

Class: **SensitivityDlgJ**

Member: `m_initialValue : double`

Description: When a text field gets the focus, the window text is stored here so when when the focus is lost, a comparison will indicate if a change was made.

52. **Source File:** SimTypeDlg.h

Class: **SimTypeDlg**

Function: **SimTypeDlg(CWnd* pParent = NULL) : Constructor**

Description: The user selects a simulation type from a list. The result will be in `m_sim`.


53. **Source File:** TimestepDlg.h

Class: **TimestepDlg**

Function: **TimestepDlg(TimestepDlg(StringArray &timestepList,IntervalMode mode = TimestepMode, CWnd* pParent = NULL) : Constructor**

Description: An editor for (timestep, value) pairs. `m_timestepArrayList` will have the result of the edits if OK'ed.

Input Args: `timestepList`: A flat list of (timestep, value) pairs.

Input Args: Specify the type of timesteps being edited (needed for displaying proper string descriptions).

1.0 C++ Classes for Wasp Builder by Source File

54. Source File:  WaspBuilder.h

Class:  CWaspBuilderApp
Function:  RcsFileOpen(CWaspBuilderDoc *pDoc) : BOOL
Description:  Closes pDoc (necessary to remove the file lock) and brings up the RCSOpenDlg with pDoc's file versions.
Input Args:  pDoc: pointer to the document whose pathfile will give the versions to revert to.
Returns:  TRUE if successful

Class:  CWaspBuilderApp
Function:  RcsFileSave(CWaspBuilderDoc *pDoc) : BOOL
Description:  Closes pDoc (necessary to remove the file lock) and brings up the RCSSaveDlg with pDoc's file versions.
Input Args:  pDoc: pointer to the document whose pathfile will allow the RCS dialog to show the previous versions.
Returns:  TRUE if successful

Class:  CWaspBuilderApp
Function:  EnableShellOpen() : void
Description:  Override is necessary because long filenames screw up CWinApp::RegisterShellFileTypes.

Class:  CWaspBuilderApp
Member:  m_pViewTemplate : CMultiDocTemplate*
Description:  Use this view template to instantiate new WaspBuilderViews (used by OnWaspbuilderNew).

Class:  CWaspBuilderApp
Member:  m_pReportTemplate : CMultiDocTemplate*
Description:  Use this view template to instantiate new WaspBuilderReportViews (used by OnReportNew).

Class:  CWaspBuilderApp
Member:  m_dwSplashTime :DWORD
Description:  Start time of splash screen stays up in milliseconds. Look inInitInstance to see how to change the display time (SetTimer).

Class:  CWaspBuilderApp
Member:  m_splash : SplashWindow
Description:  The Splash screen object. Look inInitInstance to see how to change the display time (SetTimer).

55. Source File:  WaspBuilderDoc.h

Class:  CWaspBuilderDoc
Member:  m_networkMode : NetworkMode
Description:  When a user interacts with the view, this member will have the current mode of operation (i.e. add a new node, etc).
55. Source File: WaspBuilderDoc.h

Class: CWaspBuilderDoc
Member: m_wc : WaspClass*
Description: Use this member anytime a WASP input file needs to parsed.

Class: CWaspBuilderDoc
Description: Dialogs for editing global card data.

Class: CWaspBuilderDoc
Member: m_pdlg : ProjectEditDlg
Description: Managing dialog of the global parameter dialogs.

Class: CWaspBuilderDoc
Member: m_programDir : CString
Description: This is the installation program directory.

Class: CWaspBuilderDoc
Member: m_readWaspPathName : CString
Description: The name of the WASP input file, used by the test read wasp thread.

Class: CWaspBuilderDoc
Member: m_basePathName : CString
Description: The base name of the WASP export file.

Class: CWaspBuilderDoc
Member: m_sensPathName : CString
Description: The base name of the sensitivity file name.

Class: CWaspBuilderDoc
Member: m_defaultFile : CString
Description: This is the current range values file.

Class: CWaspBuilderDoc
Member: m_rangeDlg : RangeDlg
Description: Allows the user to set min and max ranges for parameters and coefficients.

Class: CWaspBuilderDoc
Member: m_pSegmentArray : CWaspBuilderCntrItem**
Description: Allows for direct lookup of segments. Rebuild each time the network or a segmentID is altered.

Class: CWaspBuilderDoc
Member: m_paletteDlg : PaletteDlg
Description: The palate dialog for selecting different types of nodes to add.

Class: CWaspBuilderDoc
Member: m_showAnchorLines : BOOL
Description: Draw node anchor connections if TRUE.
1.0 C++ Classes for Wasp Builder by Source File

Class: CWaspBuilderDoc
Member: m_cardCompleteDlg : CardCompleteDlg
Description: The card completeness dialog.

Class: CWaspBuilderDoc
Member: m_impNoseg : int
Description: When importing, set these to the number of segments and systems in the import data file so that input files with different numbers of systems and segments can be imported.

Class: CWaspBuilderDoc
Member: m_nodeList : CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*> 
Description: The list of all nodes in the network.

Class: CWaspBuilderDoc
Member: m_version : int
Description: The version of the project file; useful for backward compatibility.

Class: CWaspBuilderDoc
Function: GetStartPosition() : POSITION
Description: The index of the first node in m_nodeList.

Class: CWaspBuilderDoc
Function: GetNextItem(POSITION &pos) : CWaspBuilderCntrItem*
Description: Returns CWaspBuilderCntrItem indexed by 'pos' in m_nodeList and sets 'pos' to the next element.
Input Args: pos: The index to use for dereferencing m_nodeList. pos is set to the next element upon return.

Class: CWaspBuilderDoc
Function: RemoveItem(CWaspBuilderCntrItem *pRemoveItem) : BOOL
Description: Removes 'pRemoveItem' from m_nodeList.
Returns: TRUE if successful.

Class: CWaspBuilderDoc
Member: m_labelFont : CFont
Description: Font to draw all future labels in.

Class: CWaspBuilderDoc
Function: SetDefaultFont(CWaspBuilderCntrItem* pLabelItem) : void
Description: Assigns m_labelFont from the label item to the default m_labelFont.
Input Args: pLabelItem: The label item to get the default from.

Class: CWaspBuilderDoc
Function: RegisterWASPControls() : void
Description: Register all OCXs used by the application (unused).

Class: CWaspBuilderDoc
Function: GetHitItem(CPoint point) : CWaspBuilderCntrItem*
Description: Intersect a point with every object in the network and return first one found.
Class: **CWaspBuilderDoc**
Function: CreateWASPNet() : void
Description: Build a network based on an input file (m_wc).

Class: **CWaspBuilderDoc**
Function: DeleteNetwork() : void
Description: Remove all nodes from the network.

Class: **CWaspBuilderDoc**
Function: GetGroupEBreaks(CArray<int, int> &nbbreaksArray) : BOOL
Description: Return the number of breaks in the time functions for Group E by querying the number of breaks in each segment control. Each array element is the number of breaks for the particular system.
Input Args: nbbreaksArray: One element for each system; each element is the number of breaks in the system.
Returns: TRUE if successful.

Class: **CWaspBuilderDoc**
Function: SetGroupEBreaks(CArray<int, int> &nbbreaksArray) : void
Description: Initialize space for all systems in group E so all segments have the same number of breaks for that system.
Input Args: nbbreaksArray: One element for each system; each element is the number of breaks in the system.

Class: **CWaspBuilderDoc**
Function: SetGroupEBreaks(int whichSystem, CArray<int, int> &nbbreaksArray) : void
Description: Initialize space for the given system in group E so all segments have the same number of breaks for that system.
Input Args: whichSystem: the system of interest.
Input Args: nbbreaksArray: One element for each system; each element is the number of breaks in the system.

Class: **CWaspBuilderDoc**
Function: GetGroupESegmentList(int whichSystem, CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>* &segList) : void
Description: Get a list of segments that have group E data.
Input Args: whichSystem: the system of interest.
Input Args: segList: Output parameter with the list of segments.

Class: **CWaspBuilderDoc**
Function: GetGroupFSegmentList(int whichSystem, CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>* &segList) : void
Description: Get a list of segments that have group F data.
Input Args: whichSystem: the system of interest.
Input Args: segList: Output parameter with the list of segments.
1.0 C++ Classes for Wasp Builder by Source File

Class: CWaspBuilderDoc
Function: GetGroup1TimeFuncList(Chist<int, int> &timefuncList) : void
Description: Get the number of time functions defined by the project.
Input Args: timefuncList: A list of indices of time functions that have data.

Class: CWaspBuilderDoc
Function: WriteGroup[A-J](FILE *fp, variable arguments) : BOOL
Description: Output the card to file 'fp'
Input Args: fp: The file pointer to write to.
Returns: TRUE if successful.

Class: CWaspBuilderDoc
Function: ReadGroup[A-J](FILE *fp, BOOL dontStore = TRUE) : BOOL
Description: Read the card from file 'fp'. If dontStore is TRUE, then just test if the card parses correctly (used to import cards later on in the input file).
Input Args: fp: The file pointer to read from.
Input Args: dontStore: If FALSE, then update state; otherwise just parse the card.
Returns: TRUE if successful.

Class: CWaspBuilderDoc
Function: ImportCheck(BOOL &isCompleteInput) : FILE*
Description: Open the file and attempt to read as a WASP input file. Return non-NULL if either the file passes a sanity check by comparing with the group A dialog members or no group A data is read, in which case the calling function is left to parse the file for its particular card data. The file pointer will be left at the start of the either group B (if the input file is a complete WASP input set) or the beginning (if group A data not found). Set isCompleteInput to TRUE if card A is read.
Input Args: isCompleteInput: TRUE if the input file contains a card A (otherwise the file is assumed to just contain the individual card).
Returns: TRUE if successful.

Class: CWaspBuilderDoc
Function: GetSegmentPtrFromID(int id) : CWaspBuilderCntrItem*
Description: Return the segment node withID 'id'.
Input Args: id: the 1 based index of the segment.
Returns: TRUE if successful.

Class: CWaspBuilderDoc
Function: AddSegment(CWaspBuilderCntrItem* pSegItem) : void
Description: Updates data structures for a new segment (added at the end of the segment list).
Input Args: pSegItem: The segment to add.
Returns: TRUE if successful.
Class: CWaspBuilderDoc
Function: InitializeSegment(CWaspBuilderCntrItem* pSegItem, int segIdx) : void
Description: Initializes the segment node from m_wc. Simplifies calling the individual segment functions.
Input Args: pSegItem: The segment to initialize.
Input Args: segIdx: The index of the segment to reference in m_wc.

Class: CWaspBuilderDoc
Function: SetGroupC_SegmentData(CWaspBuilderCntrItem* pSegItem) : void
Description: Sets group C data in every segment using m_wc.
Input Args: pSegItem: The segment to initialize.

Class: CWaspBuilderDoc
Function: InitializeExchange(CWaspBuilderCntrItem* pExchItem) : void
Description: Sets exchange data (cards B and D) data in given exchange using m_wc.
Input Args: pExchItem: The exchange to initialize.

Class: CWaspBuilderDoc
Function: SetGroupB_NTEX(CWaspBuilderCntrItem* pExchItem, int new_ntex = -1) : void
Description: Sets group B NTEX data data the given exchange using m_wc. If default args are given,
query the group B global dialog for the values.
Input Args: pExchItem: The exchange to initialize.
Input Args: new_ntex: The new NTEX value.

Class: CWaspBuilderDoc
Function: SetGroupB_NTEX(GroupName pore_or_surfwater = SURFWATER, int new_ntex = -1) : void
Description: Sets group B NTEX data data in every exchange using m_wc. If default args are given,
query the group B global dialog for the values.
Input Args: pore_or_surfwater: The field to set NTEX for.
Input Args: new_ntex: The new NTEX value.

Class: CWaspBuilderDoc
Function: SetGroupB_ExchangeData(CWaspBuilderCntrItem* pExchItem) : void
Description: Assign all exchange info from m_wc to the given exchange item.
Input Args: pExchItem: The exchange item to assign to.

Class: CWaspBuilderDoc
Function: SetGroupD_Flows() : void
Description: Assign flow data to every exchange node.

Class: CWaspBuilderDoc
Function: SegIDChanged(CWaspBuilderCntrItem* pChangedItem, int oldID) : void
Description: If the user changes the ID of a segment, then there must be a reordering of segments so
that numbering 1..n is preserved. This will pop up a dialog to get user preferences for reordering.
1.0  C++ Classes for Wasp Builder by Source File

Class:  CWaspBuilderDoc
Function:  UpdateProjectParams(CWaspBuilderCtrItem* pChangedItem) : void
Description:  Update global data whose structure depends on nodes. For example, if a segmentID is
altered, then the segment array will need to be recreated.

Class:  CWaspBuilderDoc
Function:  ReadWasp(CString pathname) : void
Description:  Initialize the m_wc member.

Class:  CWaspBuilderDoc
Function:  WriteWasp() : CString
Description:  Write network as WASP input file and return the file name.
Returns:  The new WASP file.

Class:  CWaspBuilderDoc
Function:  ExecuteWasp(CString fname) : BOOL
Description:  Run the appropriate model with input file 'fname'.
Returns:  TRUE if successful.

Class:  CWaspBuilderDoc
Function:  Locatewasp5executables() : CString
Description:  Ask the user to locate the WASP5 executables tox5.exe and eutro5.exe.
Returns:  The directory with the WASP5 model executables.

Class:  CWaspBuilderDoc
Function:  GenerateSensitivityRun(CardEnums whichCard, CString *pathname) : CString
Description:  Ask the user for the directory and basename to store the sensitivity analysis runs under.
    Builds the static portions of the input called <basename>_pre.inp and <basename>_post.inp. ExecuteSensitivityAnalysis will merge the static portions and
    the card that is being analyzed.
Returns:  The base name of the WASP input file.

Class:  CWaspBuilderDoc
Function:  ExecuteSensitivityAnalysis(CString pathname, CString fname, int interval, BOOL
    useBasecase=TRUE) : void
Description:  Merge the static and the portion of the input that has the card that is being studies. Run
    the model for each input file generated. Pops up the OutputDlg when finished.
Input Args:  pathname: The directory to use for input file generation.
Input Args:  fname: The base name of the WASP input file set.
Input Args:  interval: The number of runs to create.
Input Args:  useBasecase: if TRUE, then run the basecase as well.

Class:  CWaspBuilderDoc
Function:  BuildSegmentArray() : void
Description:  Updates m_pSegmentArray.
Class: CWaspBuilderDoc
Function: DeleteNode(CWaspBuilderCntrItem* pDelItem) : void
Description: Delete the node given by 'pDelItem'. May cause a reshuffling event to occur so that
ordering 1..n is preserved.

Class: CWaspBuilderDoc
Function: CheckGlobalDataCompleteness(CStringList &linelist) : void
Description: Return a list of possible holes in global data.
Input Args: linelist: Output argument with the results of the query.

Class: CWaspBuilderView
Function: CWaspBuilderView() : Constructor
Description: Dialogs for editing global card data.

Class: CWaspBuilderView
Member: m_pSelection : CWaspBuilderCntrItem*
Description: The currently selected node.

Class: CWaspBuilderView
Member: m_curPoint : CPoint
Description: The current location of the mouse. This is used to display different cursors over nodes
and empty space.

Class: CWaspBuilderView
Function: DrawConnection(CDC* pDC, CWaspBuilderCntrItem* topNode,
CWaspBuilderCntrItem* botNode, BOOL drawArrowhead = TRUE) : void
Description: Draw a connecting line between an upstream and downstream node.
Input Args: topNode: End target of line draw.
Input Args: botNode: Start target of the line draw.
Input Args: drawArrowhead: Put on arrowhead (-->) on the top of the line.

Class: CWaspBuilderView
Function: GetHitItem(CPoint point, int *hit_test = NULL) : CWaspBuilderCntrItem*
Description: Test if the point intersects a node. Return the node if successful; otherwise return
NULL.
Input Args: point: Point to test.
Input Args: hit_test: Result of CRectTracker::HitTest
Returns: The intersecting node if successful; otherwise NULL.

Class: CWaspBuilderView
Function: GetHitItem(CRect checkRect) : CWaspBuilderCntrItem*
Description: Intersect a rectangle with every network item. Return the first hit object if successful.
Input Args: checkRect: Intersecting rectangle.
Returns: The intersecting node if successful; otherwise NULL.
Class: **CWaspBuilderView**
Function: `DocToClient(CRect& rect) : void`
Description: Translate the scrolled rect 'rect'(doc view) to window coords (client view).
Input Args: `rect`: The rect to convert.

Class: **CWaspBuilderView**
Function: `ClientToDoc(CRect& rect) : void`
Description: Translate the window rect 'rect' (client view) to scrolled win coords (client view).
Input Args: `rect`: The rect to convert.

Class: **CWaspBuilderView**
Function: `ClientToDoc(CPoint& point) : void`
Description: Translate the window point 'point' (client view) to scrolled win coords (client view).
Input Args: `point`: The point to convert.

Class: **CWaspBuilderView**
Function: `GetScrolledItemRect(CWaspBuilderCntrItem* pItem) : CRect`
Description: Calculate the placement of the node on the scroll view.
Input Args: `pItem`: The node to place in scrolled view coordinates.
Returns: The location of the item in view screen coordinates.

Class: **CWaspBuilderView**
Function: `SetupTracker(CRectTracker* pTracker, CWaspBuilderCntrItem* pItem, CRect* pTrueRect = NULL) : void`
Description: Initialize pTracker state from 'pItem' (i.e. if the node is selected, then give the tracker resize handles).
Input Args: `pItem`: The node to build a tracker for.

Class: **CWaspBuilderView**
Function: `InvalidateItem(CWaspBuilderCntrItem* pItem) : void`
Description: Redraw 'pItem'.
Input Args: `pItem`: The node to redraw.

Class: **CWaspBuilderView**
Function: `SetSelection(CWaspBuilderCntrItem* pNewSel, BOOL bSafeSelect = FALSE) : void`
Description: Assign m_pSelection to 'pNewSel'. Redraw both the old selection and the new so that they reflect their new state.
Input Args: `pItem`: The node to select.

Class: **CWaspBuilderView**
Function: `CWaspBuilderCntrItem* ConvertIDToPtr(CStringID) : void`
Description: Get the segment node that represents 'ID'.
Input Args: `id`: 1..n based segment index.

Class: **CWaspBuilderView**
Function: `ConvertIDToPtr(CStringList &IDList, CList<CList<CWaspBuilderCntrItem*, CWaspBuilderCntrItem*>&& ptrList) : void`
Description: Convert a list of stringIDs to container pointers. Used to get handles to anchored nodes.
Class: **CWaspBuilderView**
Function: NewExchange(CWaspBuilderCtrItem* downstreamnode, ExchModeEnum
type=EXCH_MODE_NONE) : void
Description: Connect the current selection (m_pSelection) to the argument node.
Input Args: downstreamnode: Target of new exchange.
Input Args: type: Type of exchange to create (see ExchModeEnum).

Class: **CWaspBuilderView**
Function: MakeAnchor(CWaspBuilderCtrItem* anchornode) : void
Description: Anchor the selection (m_pSelection) to 'anchornode'. Any time 'anchorNode' is
moved, the anchored node will move the same amount.
Input Args: anchornode: Target of anchor operation

Class: **CWaspBuilderView**
Function: MakeAnchor(CWaspBuilderCtrItem* anchorree) : void
Description: Sever the selection node from its connection to 'anchorree'.
Input Args: anchorree: Target of unanchor operation.

Class: **CWaspBuilderView**
Function: BuildAnchorList(CWaspBuilderCtrItem* sourceItem,
CList<CWaspBuilderCtrItem*, CWaspBuilderCtrItem*> &anchorListPtr) : void
Description: Recursively descend the source item to build a list of all nodes anchored to the item.
Input Args: sourceItem: Node to query for anchored nodes.
Input Args: anchorListPtr: Store pointer to every anchored node here.

Class: **CWaspBuilderView**
Function: ScrollDrag(CPoint curLocation) : void
Description: Scroll the view if a drag operation goes off the view.
Input Args: curLocation: Point indication direction to scroll. If x < 0, scroll up, etc.

Class: **CWaspBuilderView**
Member: m_groupNodeList : CList<CWaspBuilderCtrItem*, CWaspBuilderCtrItem*>>
Description: List of all nodes that have grouped together.

57. **Source File:** WaspClass.h

Class: **WaspClass**
Function: WaspClass() : Constructor
Description: Class for reading and writing WASP input data.

58. **Source File:** WASPReportDoc.h

Class: **CWASPReportDoc**
Function: CWASPReportDoc() : Constructor
Description: Document for the report view.
Class: **CWASPReportDoc**
Function: **AddFile** (CString pathName, BOOL addAtEnd=TRUE) : BOOL
Description: Add 'pathName' to the list of WASP input files.
Input Args: 
- **pathName**: The file name of the new WASP input file.
- **addAtEnd**: Append to the list; otherwise add at beginning.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Function: **AddNode** (int ID) : BOOL
Description: Add a new segment (node) 'ID' to the list of segments.
Input Args: 
- **ID**: the 1..n based segmentID number.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Function: **DeleteFile** (int pos) : BOOL
Description: Remove the file with index 'pos' from the list of WASP input files.
Input Args: 
- **ID**: the 0 based index of the file in m_waspOutputData.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Function: **DeleteNode** (int pos) : BOOL
Description: Remove the segment (node) with index 'pos' from m_parameterData.
Input Args: 
- **ID**: the 0 based index of the segment.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Function: **GetMin** (BOOL useTRN, int whichVar, double *retVal) : BOOL
Description: Retrieve the parameter min for all the output data files.
Input Args: 
- **useTRN**: if TRUE, then query the transient output parameters; otherwise query the transient simulation parameters.
- **retVal**: Store the minimum result here.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Function: **GetMax** (BOOL useTRN, int whichVar, double *retVal) : BOOL
Description: Retrieve the parameter max for all the output data files for the given type (transient (TRN) or sim).
Input Args: 
- **useTRN**: if TRUE, then query the transient output parameters; otherwise query the simulation parameters.
- **retVal**: Store the maximum result here.
Returns: TRUE if successful.

Class: **CWASPReportDoc**
Member: **m_parameterData** : ParameterData
Description: The parameter and segment information.
Class: CWASPReportDoc
Member: m_waspOutputData : CArray<WaspOutputData, WaspOutputData&>
Description: A list of all WASP input filenames

59. Source File: waspreportview.h

Class: CWASPReportView
Function: CWASPReportView() : Constructor
Description: View class for report window.

Class: CWASPReportView
Function: UpdateFilenameTable() : void
Description: Repopulate the filename list.

Class: CWASPReportView
Function: UpdateNodeTable() : void
Description: Repopulate the segment list.

Class: CWASPReportView
Function: UpdateParameterList() : void
Description: Repopulate the parameter list box. This will include all output parameters, both transient and simulation.

Class: CWASPReportView
Function: GetColsPerNode() : int
Description: The number of columns of data that will be output for each segment (based on report option toggles).
Returns: The number of columns of data that will be output for each segment

Class: CWASPReportView
Function: BuildReportColumns() : void
Description: Create columns for the report data based on report options and selected parameters.

Class: CWASPReportView
Function: UpdateReportTable() : void
Description: Populate the report table with information for selected parameters and all the segments and files.

60. Source File: WaspUtils.h

Class: None
Function: SaveRecCommonFmt(FILE *fp, StringArray &timestepArray) : void
Description: Write the timestep data in 'timestepArray' to file 'fp' as a list of (val, day) pairs suitable for reading by WASP.
Input Args: fp: The file to write to.
Input Args: timestepArray: Write this interval data to the file.
Returns: TRUE if the timestepArray parsed correctly.
1.0 C++ Classes for Wasp Builder by Source File

Class: None
Function: ReadRecCommonFmt(FILE *fp, StringArray &timestepArray, int nobrk=-1) : void
Description: Write the timestep data in 'timestepArray' to file 'fp' as a list of (val, day) pairs suitable for reading by WASP.
Input Args: fp: The file to write to.
Input Args: timestepArray: Write this interval data to the file.
Returns: TRUE if the timestepArray parsed correctly.
# 2.0 Function Index Arranged by Class

## A

### CWaspBuilderDoc
- ReadGroup **53**
- WriteGroup **53**

## B

### BypassOptsDlg
- BypassOptsDlg **1**

## C

### CardCompleteDlg
- CardCompleteDlg **1**
- UpdateAll **1**

### CWaspBuilderApp
- EnableShellOpen **49**
- RcsFileOpen **49**

### CWaspBuilderCntrItem
- Anchor **4**
- CWaspBuilderCntrItem **2**
- Delete **4**
- DeleteItemsDlg **4**
- Draw **2**
- GetMidPoint **2**
- GetNodeType **3**
- GetParameter **3**
- GetRect **2**
- GetSize **2**
- Invalidate **2**
- IsGroup **4**
- Move **2**
- ParameterChanged **3**
- PopupActionWindow **3**
- SegmentJuxtapose **3**
- Serialize **4**
- SetData **4**
- SetNodeType **3**
- SetParameter **3**
- SetRect **2**
- SetSize **2**
- UpdateExtent **3**

### CWaspBuilderDoc
- AddSegment **53**
- BuildSegmentArray **55**
- CheckGlobalDataCompleteness **56**
- CreateWASPNet **52**
- DeleteNetwork **52**
- DeleteNode **56**
- ExecuteSensitivityAnalysis **55**
- ExecuteWasp **55**
- GenerateSensitivityRun **55**
- GetGroupEBreaks **52**
- GetGroupESegmentList **52**
- GetGroupFSegmentList **52**
- GetGroupITimefuncList **53**
- GetHitItem **51**
- GetNextItem **51**
- GetSegmentPtrFromID **53**
- GetStartPosition **51**
ImportCheck 53
InitializeExchange 54
InitializeSegment 54
Locatewasp5executables 55
ReadWasp 55
RegisterWASPControls 51
RemoveItem 51
SegIDChanged 54
SetDefaultFont 51
SetGroupB_ExchangeData 54
SetGroupB_NTEX 54
SetGroupC_SegmentData 54
SetGroupD_Flows 54
SetGroupEBreaks 52
UpdateProjectParams 55
WriteWasp 55

CWaspBuilderView
BuildAnchorList 58
ClientToDoc 57
ConvertIDToPtr 57
CWaspBuilderCntrlItem 57
CWaspBuilderView 56
DocToClient 57
DrawConnection 56
GetHitItem 56
GetScrolledItemRect 57
InvalidateItem 57
MakeAnchor 58
NewExchange 58
ScrollDrag 58
SetSelection 57
SetupTracker 57

CWASPRReportDoc
AddFile 59
AddNode 59
CWASPRReportDoc 58
DeleteFile 59
DeleteNode 59
GetMax 59
GetMin 59

CWASPRReportView
BuildReportColumns 60
CWASPRReportView 60

GetColsPerNode 60
UpdateFilenameTable 60
UpdateNodeTable 60
UpdateParameterList 60
UpdateReportTable 60

E

ExchangeNodeDlg
ExchangeNodeDlg 4
GetChangedParametersArray 5
getParameter 5
GraphControlDlg 5
Serialize 5
SetParameter 5

G

GraphDlg
~GraphDlg 6
AdvanceFrame 12
DrawGraph 6
GraphDlg 5
SetDimensions 6
SetFrameData 6
SetTitles 6
StartAnimation 11

GroupADlg
CheckGlobalDataCompleteness 13, 14
GetMaxNoSys 12
GroupADlg 12
GroupBDlg 13
LoadFromWC 12, 13
ReadAR 12, 13
UpdateJMASS 13
WriteAR 12, 14
WriteInput 12

GroupBNodeDlg
GetChangedParametersArray 14
getParameter 14
2.0 Function Index Arranged by Class

GroupBNodeDlg
SetParameter 14
UpdateTable 15

GroupCDlg
CheckGlobalDataCompleteness 16
GroupCDlg 15
LoadFromWC 16
ReadAR 14
WriteAR 16

GroupCNodeDlg
GetChangedParametersArray 17
GetParameter 16
GroupCNodeDlg 16
GroupDDlg 17
SetParameter 16
UpdateTable 17

GroupDDlg
CheckGlobalDataCompleteness 18
LoadFromWC 17
NewTimefunc 18
ReadAR 17
UpdateSymType 18
WriteAR 17

GroupDNodeDlg
GetChangedParametersArray 19
GetParameter 16
GroupDNodeDlg 18
SetParameter 19
UpdateTable 19

GroupEDlg
CheckGlobalDataCompleteness 20
GroupEDlg 20
LoadFromWC 20
ReadAR 20
UpdateSymType 20
WriteAR 20

GroupENodeDlg
GetChangedParametersArray 21
GetParameter 21
GroupENodeDlg 21
SetParameter 21
UpdateSymType 21
UpdateTable 21

GroupFDlg
CheckGlobalDataCompleteness 23
GetSegIdxFromID 22
GroupFDlg 22
InitSegmentArray 23
InitSystemListItem 22
LoadFromWC 22
ReadAR 22
SetSegData 23
UpdateSymType 23
WriteAR 22

GroupFNodeDlg
GetChangedParametersArray 24
GetParameter 24
GroupFNodeDlg 24
SetParameter 24
UpdateSymType 24
UpdateTable 24

GroupGDlg
CheckGlobalDataCompleteness 25
GroupGDlg 25
LoadFromWC 25
ReadAR 25
RedrawTable 25
SetNParams 25
UpdateHelpLabel 25
WriteAR 25

GroupGNodeDlg
GetChangedParametersArray 26
GetParameter 26
GroupGNodeDlg 26
SetParameter 26
UpdateSymType 26
UpdateTable 27
GroupHDlg
  AllocateDatafieldMemory 28
  AllocateSystemMemory 28
  CheckGlobalDataCompleteness 28
  GroupHDlg 27
  LoadFromWC 27
  ReadAR 27
  UpdateConstTable 28
  UpdateFieldTable 28
  UpdateSymType 28
  WriteAR 28

GroupIDlg
  CheckGlobalDataCompleteness 30
  GroupIDlg 29
  LoadFromWC 29
  ReadAR 29
  SetNumTimefuncs 29
  UpdateHelpLabel 29
  WriteAR 29

GroupJDlg
  AddSegment 31
  AddSystem 31
  CheckGlobalDataCompleteness 31
  GroupJDlg 30
  LoadFromWC 30
  ReadAR 30
  UpdateSymType 31
  WriteAR 30

InputPlotCDlg
  InputPlotCDlg 32

InputPlotDDlg
  GetWaterFlowLabel 32
  InputPlotDDlg 32

InputPlotEDlg
  InputPlotEDlg 32

InputPlotFDlg
  InputPlotFDlg 32

InputPlotIDlg
  InputPlotIDlg 33
  SetNumTimefuncs 33

LabelNodeDlg
  GetChangedParametersArray 33
  GetFont 33
  GetParameter 34
  LabelNodeDlg 33
  Serialize 34
  SetFont 33
  SetParameter 34

MultiTrack
  MultiTrackerDrawTrackerRect 34

NodeData
  AddNode 38
  DeleteNode 38
  NodeData 38
  NodeOutputData 39
  ParameterData 38

HasData 19

InputError
  ErrorMsg 43
  InputError 42

InputPlotBDlg
  InputPlotBDlg 31
2.0 Function Index Arranged by Class

NodeOutputData
- GetMax 40
- GetMin 39
- VariableOutputData 39

None
- ReadRecCommonFmt 61
- SaveRecCommonFmt 60

O

OutputDlg
- CheckDiffSensButton 35
- GetOutputTypeAndExtName 34
- InitializeSegAndVarList 35
- InitializeTypeCombo 34
- OutputDlg 34
- ReadFiles 35
- UpdateOutputView 35

P

PaletteDlg
- PaletteDlg 35
- SetButtonSensitivity 35

R

RangeDlg
- ~RCSClass 36
- CIVersion 36
- COVersion 36
- DeleteVersion 36
- GetFileName 36
- InitData 36
- RangeDlg 35
- RCSClass 36
- SaveTableData 36
- SetFileName 36
- UpdateTableData 36

RcsFileOpen
- RcsFileSave 49

RCSOpenDlg
- InitGui 37
- RCSOpenDlg 37

RCSSaveDlg
- InitGui 37
- RCSSaveDlg 37

S

SegmentChooserDlg
- SegmentChooserDlg 43

SegmentInsertDlg
- SegmentInsertDlg 43

SegmentNodeDlg
- GetChangedParametersArray 43
- GetParameter 44
- SegmentNodeDlg 43
- Serialize 44
- SetParameter 44
- UpdateParameters 44
- UpdateSymType 44

SensAnalysisParamsDlg
- SensAnalysisParamsDlg 44

SensitivityDlgB
- SensitivityDlgB 44
- UpdateCurVal 45
- UpdateValuesList 45

SensitivityDlgC
- SensitivityDlgC 45
- UpdateCurVal 45
- UpdateValuesList 45

SensitivityDlgD
- SensitivityDlgD 45
- UpdateCurVal 46
- UpdateValuesList 46
SensitivityDlgE
  SensitivityDlgE 46
  UpdateCurVal 46
  UpdateValuesList 46

SensitivityDlgF
  SensitivityDlgF 46
  UpdateCurVal 46
  UpdateValuesList 46

SensitivityDlgG
  LoadParameterCombo 47
  SensitivityDlgG 47
  UpdateCurVal 47
  UpdateValuesList 47

SensitivityDlgH
  SensitivityDlgH 47
  UpdateCurVal 47
  UpdateValuesList 48

SensitivityDlgJ
  SensitivityDlgJ 48
  UpdateCurVal 48
  UpdateValuesList 48

SimTypeDlg
  SimTypeDlg 48
2.0 Member Index Arranged by Class

A

CWaspBuilderDoc
 m_ 50

B

BypassOptsDlg
 m_bypassOptsArrayList 1
 m_systemCheckList 1

C

CWaspBuilderApp
 m_dwSplashTime 49
 m_pReportTemplate 49
 m_pViewTemplate 49
 m_splash 49

CWaspBuilderDoc
 m_basePathName 50
 m_cardCompleteDlg 51
 m_defaultFile 50
 m_impNoseg 51
 m_labelFont 51
 m_networkMode 49
 m_nodeList 51
 m_paletteDlg 50
 m_pdlg 50
 m_programDir 50
 m_pSegmentArray 50
 m_rangeDlg 50
 m_readWaspPathName 50
 m_sensPathName 50
 m_showAnchorLines 50
 m_version 51
 m_wc 50

CWaspBuilderView
 m_curPoint 56
 m_groupNodeList 58
 m_pSelection 56

CWASPReportDoc
 m_parameterData 59
 m_waspOutputData 60

D

dlg 50

E

ExchangeNodeDlg
 m_changedParameters 5
GraphDlg
  m_backgroundColor 10
  m_colors 7
  m_currentFrame 7
  m_dataLabels 9
  m_doNotDelete 11
  m_graphStyle 7
  m_graphTitle 10
  m_graphType 6
  m_labels 9
  m_labelXDateStart 9
  m_legendPos 10
  m_linePattern 7
  m_nCols 9
  m_nFrames 7
  m_nLabels 9
  m_nOverlaySets 8
  m_nSets 7
  m_overlayColors 7
  m_overlayGraphStyle 8
  m_overlayGraphType 8
  m_overlaySetData 8
  m_overlaySetMissing 8
  m_overlaySetTitle 8
  m_overlayTitle 8
  m_overlayTitleStyle 8
  m_setData 7
  m_setDist 7
  m_setTitle 7
  m_showYAxisTicksLeft 9
  m_showYAxisTicksRight 10
  m_size 6
  m_useDistData 8
  m_useGridX 11
  m_useGridY 11
  m_useOverlayDistData 9
  m_windowTitle 10
  m_xAxisMin 11
  m_XAxisTicks 9
  m_XAxisTicksMinor 9
  m_XTitle 10
  m_yAxisMaxLeft 11
  m_yAxisMaxRight 11
  m_yAxisMinLeft 11
  m_yAxisMinRight 11
  m_YAxisTextLeft 10
  m_YAxisTextRight 10
  m_YAxisTicksLeft 9
  m_YAxisTicksMinorLeft 9
  m_YAxisTicksMinorRight 10
  m_YAxisTicksRight 10
  m_YTitle 10

GroupADlg
  m_iseqoutArrayList 13
  m_printIntervalArrayList 13
  m_systemBypassArrayList 13
  m_timestepArrayList 13

GroupBDlg
  m_porewaterArrayList 14
  m_rbykArrayList 14
  m_surfwaterArrayList 14

GroupBNodeDlg
  m_backDataArray 15
  m_changedParameters 15
  m_dataArray 15
  m_pore_ntex 15
  m_surf_ntex 15

GroupCNodeDlg
  m_changedParameters 17

GroupDDlg
  m_currentSystem 18
  m_flowArrayLists 18
  m_systemBypassArrayList 18

GroupDNodeDlg
  m_changedParameters 19
  m_flowFieldArray 19
  m_nmaxtimefuncs 19
2.0 Member Index Arranged by Class

**GroupEDlg**
- m_boundArrayList 20
- m_nbreaksArrayboundArrayList 20

**GroupENodeDlg**
- m_boundaryListArray 21
- m_changedParameters 22

**GroupFDlg**
- m_backSystemListArray 23
- m_currentSystem 23
- m_systemListArray 23

**GroupFNodeDlg**
- m_changedParameters 24
- m_systemListArray 24

**GroupGDlg**
- m_nparams 26
- m_paramValueList 26
- m pnameValueList 26

**GroupGNodeDlg**
- m_changedParameters 27
- m_descripArray 27
- m_nParams 27
- m_paramArray 27

**GroupHDlg**
- m_backupListArray 28
- m_currentField 29
- m_currentSystem 29

**GroupIDlg**
- m_aname 30
- m_ntimefuncs 30
- m_timefunc 30

**GroupIDlg**
- m_backupCListArray 31
- m_currentSystem 31
- m_ICListArray 31

**InputPlotBDlg**
- m_exchList 32
- m_waterArrayList 32

**InputPlotDDlg**
- m_exchList 32

**InputPlotIDlg**
- m_ntimefuncs 33
- m_validTimefuncsList 33

**LabelNodeDlg**
- m_changedParameters 34

**NodeData**
- m_parameterArray 38

**NodeOutputData**
- m_nodeOutputData 40
- m_periodOutputData 39
- m_varName 40
- m_varUnits 40

**ParameterData**
- m_nodeData 39
- m_parameterNamesArray 39
R

RCSClass
  m_dateList 37
  m_description 37
  m_logMessageList 37
  m_symNameList 37
  m_versionList 37

S

SegmentChooserDlg
  m_additionalSegments 43
  m_changedParameters 44
  m_possibleSegments 43

SensitivityDlgB
  m_exchList 45
  m_initialValue 45
  m_waterArrayList 45

SensitivityDlgC

SensitivityDlgD
  m_initialValue 45

SensitivityDlgE
  m_initialValue 46

SensitivityDlgF
  m_initialValue 47

SensitivityDlgG
  m_initialValue 47
  m_parameterList 47

SensitivityDlgH
  m_initialValue 48

SensitivityDlgJ
  m_initialValue 48

W

WaspOutputData
  m_periodSIMArray 42
  m_periodTRNArray 42
  m_variableOutputSIMData 42
  m_variableOutputTRNData 42