Overview

Import plugins are modules used by MyInfo to import topics & documents from third-party file formats or generated on user input. To create import plugin you have to write dynamic-link library (DLL) which to include several functions, depending on the type of the plugin you want to create.

There are two types of import plugins you can use in MyInfo:

Plugins associated with file format

These plugins are bound to a particular file format, so when the user want to import data using such plugin, MyInfo shows him File Open dialog asking him for the file, he wants to import.

To create such plugin you need to write the following functions:

1. GetPluginInfo with the following options in TPluginInfo variable:
   - PI_IMPORT_DATA in *ImportCaps*.
   - Appropriate import text file format in *ImportTextFormat*. See GetPluginInfo for details.
   - File extension of the import files (the same file extension is used for export, if the plugin is combined) in *FileExt*. See GetPluginInfo for details.
   - File filter, which will be used in the File Open dialog in *FileExtDescription*. See GetPluginInfo for details.
2. ConfigurePlugin [optional]
3. BeginImportSession
4. GetAvailableCustomColumns [optional]
5. ImportCustomColumn [optional]
6. ImportDocument
7. EndImportSession

Plugins without associated file format

These plugins either show their own dialog to the user when BeginImportSession function is called or import their data from a common location (for example importing Internet Explorer Favorites does not require user interaction, because this data is stored in common location for the current user).

Such plugin must include the following the functions:

1. GetPluginInfo with the following options in TPluginInfo variable:
   - PI_IMPORT_DATA in *ImportCaps*.
   - Appropriate import text file format in *ImportTextFormat*. See GetPluginInfo for details.
   - Empty string in *FileExt* (it is empty by default).
   - Empry string in *FileExtDescription* (it is empty by default).
2. ConfigurePlugin [optional]
3. BeginImportSession
4. GetAvailableCustomColumns [optional]
5. ImportCustomColumn [optional]
6. ImportDocument
7. EndImportSession

Installation

- For MyInfo 6.10 and newer:
  To install a plugin, just place it in its own folder under the "`[Documents and Settings]\[User Name]\Application Data\Milenix\MyInfo\4\Plugins`" folder or use the installation feature from within MyInfo (*Tools* > *Add-ins* > *Plugins* > *Install*).
- For MyInfo 6.09 and older:
  To install a plugin, just place it in the "`[MyInfo folder]\Configuration\Plugins`" folder.
- If you want to install your plugin using installation program, you can obtain Plugins folder location reading the PluginsDir value in the "`HKEY_LOCAL_MACHINE\SOFTWARE\Milenix\MyInfo\6`" key on the target computer.

Import Process

When importing data from import plugin, MyInfo calls the functions in your plugin in the following order, depending on whether the plugin is associated with file format:

Plugins associated with file format

1. BeginImportSession is called, so you can open the file (stored in *file* variable, which the user has selected to import in the File Open dialog.
2. If your plugin contains GetAvailableCustomColumns, it is called.
   Use this function to check if some or all of the custom columns you want to use are already created.
3. If your plugin contains ImportCustomColumn function, it is called.
   Use this function to define custom tree columns, which you will use when importing document.
   This function will be called as long as it returns RE_SUCCESS. When there are not more columns to import, return RE_IMPORT_COMPLETED (so if you pass custom column information when returning RE_IMPORT_COMPLETED, it will be ignored).
4. ImportDocument is called. Use this function to pass the information about the first document you want to import in MyInfo.
   This function will be called as long as it returns RE_SUCCESS. When there are no more documents to import, return RE_IMPORT_COMPLETED (so if you pass document information when returning RE_IMPORT_COMPLATED, it will be ignored).
5. EndImportSession is called. Here you should free all data that you have used during import.

Plugins without associated file format

1. BeginImportSession is called. Here if your plugin needs import data location, you should display dialog, where the user can select import data source. Because your plugin is not associated with file format *file* variable will be empty.
2. If your plugin contains GetAvailableCustomColumns, it is called.
   Use this function to check if some or all of the custom columns you want to use are already created.
3. If your plugin contains ImportCustomColumn function, it is called.
   Use this function to define custom tree columns, which you will use when importing document.
   This function will be called as long as it returns RE_SUCCESS. When there are not more columns to import, return RE_IMPORT_COMPLETED (so if you pass custom column information when returning RE_IMPORT_COMPLATED, it will be ignored).
4. ImportDocument is called. Use this function to pass the information about the first document you want to import in MyInfo.
   This function will be called as long as it returns RE_SUCCESS. When there are no more documents to import, return RE_IMPORT_COMPLETED (so if you pass document information when returning RE_IMPORT_COMPLATED, it will be ignored).
5. EndImportSession is called. Here you should free all data that you have used during import.

Import Custom Columns

MyInfo allows you to create custom columns of different types (text, numbers, date and other). You can use this functionallity when importing documents in MyInfo too.

All you have to do is to include the ImportCustomColumn and GetAvailableCustomColumns functions to your plugin.

Then the process is the following:

1. MyInfo calls your GetAvailableCustomColumns function.
   Use it to get a full list of custom columns available before creating your own. Maybe some of the columns you will need are already created!
2. MyInfo calls your ImportCustomColumn function as many times, as it returns RE_SUCCESS and each time, with the data provided, it creates new custom column.
3. Now, each time when you fill the document import data in ImportDocument function, you should fill the corresponding custom column values in the TDocument *ColumnValues* variable in the same order as you created them with the *ImportCustomColumn* function, and separated by new-line character (i.e. "First custom column data\nSecond custom column data\n"). If you don't want to include some custom values for some of the documents, just fill their place with an empty new-line.

Date And Time Overview

In order to achieve multilanguage and IDE support, MyInfo has its own date and time format. It follows the International Standard ISO 8601 standard to some extent.

All date and time variables must use the following format: `YYYY-MM-DDThh:mm:ss`, where YYYY is the year (from 1900 to 2999), MM is the month (from 01 to 12), DD is the day (from 01 to 31). T indicates the beginning of the time portion of the string, where hh is the hour of the day (from 00 to 23), mm is the minute (from 00 to 59), and ss are the secounds (from 00 to 59).

Example

```
2007-07-24T14:37:59
```

This represents 24th July 2007, 14:37:59 (2:37:59 PM).

Special cases

- If the date/time variable is empty, then no date/time is set for this column in this document. You can also supply MyInfo with an empty date. In this case the program will generate the appropriate date (for example the current date and time for the DateCreated variable).
- If you want to supply a date without the time, omit the T and the following characters.

Remarks

All date/time variables are in the computer locale time zone.

GetPluginInfo Function

Provides information about the plugin such as plugin name, author, copyright notes and type.

Syntax

C/C++:

```
int __stdcall GetPluginInfo(TPluginInfo* info);
```

Delphi:

```
function GetPluginInfo(info: PPluginInfo) : integer; stdcall;
```

Parameters

info
[out] Pointer to the TPluginInfo structure that you should fill with information about the plugin.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Remarks

Plana creates and initializes *info* structure for you.

See also

TPluginInfo

ConfigurePlugin Function

Called when the user clicks Options button in About Plugins, Import or Export dialog.

Syntax

C/C++:

```
int __stdcall ConfigurePlugin(HWND callerWindow, HINSTANCE dllInstance);
```

Delphi:

```
function ConfigurePlugin(callerWindow : THandle; dllInstance : longword) : integer; stdcall;
```

Parameters

callerWindow
[in] Use this window handle to make your configuration dialog child of the caller dialog.

dllInstance
[in] You may need this DLL handle when creating your configuration dialog.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Remarks

Implement this function, only if your plugin has configuration dialog.

BeginImportSession Function

Prepares import plugin for the import process by allocating data, opening the import file or showing the user dialog, where he selects the import data location.

Syntax

C/C++:

```
int __stdcall BeginImportSession(char* fileName, HWND callerWindow, HINSTANCE dllInstance);
```

Delphi:

```
function BeginImportSession(fileName : pchar; callerWindow : THandle; dllInstance : longword) : integer;
stdcall;
```

Parameters

file
[in] Pointer to NULL-terminated string containing the full file path and name to the file you have to open (this varaible is empty string, if your plugin is not associated with file format; in this case, you have to provide the user with dialog where the user could select import data location or import data from common location).

callerWindow
[in] Use this window handle to make your select location dialog child of the caller dialog.

dllInstance
[in] You may need this DLL handle when creating your select location dialogs.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

GetAvailableCustomColumns Function

Gives you information about the custom columns available in the current MyInfo topic.

Syntax

C/C++:

```
int __stdcall GetAvailableCustomColumns(int columnCount, TCustomColumn* columns, bool reserved);
```

Delphi:

```
type TCustomColumns = array of TCustomColumn;
function GetAvailableCustomColumns(count : integer; const columns : TCustomColumns; reserved : boolean) :
integer; stdcall;
```

Parameters

columnCount
[in] Number of type integer, which tells you how many custom columns are available in the current topic. Currently, MyInfo (version 5.07) supports up to 64 columns in the topic (including the built-in).

columns
[in] Pointer to an array of TCustomColumn structures that will contain information about the available custom columns in the current topic.

reserved
[in] This value is now obsolete and no longer used.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Remarks

Do not access the columns outside the upper value of columnCount.

See also

TCustomColumn

ImportCustomColumn Function

Creates custom column in the topic tree.

Syntax

C/C++:

```
int __stdcall ImportCustomColumn(TCustomColumn* column);
```

Delphi:

```
function ImportCustomColumn(column: PCustomColumn) : integer; stdcall;
```

Parameters

column
[out] Pointer to a TCustomColumn structure that you should fill with information about the custom column you want to create.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Return RE_IMPORT_COMPLETED to indicate that there is not custom column to import and custom columns import should be ended.

Remarks

MyInfo creates and initializes *column* structure for you.

You should call this function for each custom column you want to use, even for these, which are already created in MyInfo. If you try to create an already existing custom column of the same type, MyInfo will skip it and use the already created column for importing your information.

Note, that MyInfo can have a limited number of custom columns. You can't create more columns than that limit (64 since MyInfo 4.02). Check the GetAvailableCustomColumns function's *columnCount* parameter before importing new columns to ensure that you obey this limit.

See also

TCustomColumn

ImportDocument Function

Sends information about the document you want to import.

Syntax

C/C++:

```
int __stdcall ImportDocument(TDocument* document, TMemoryRequestProc memoryRequestProc);
```

Delphi:

```
function ImportDocument(document: PDocument; memoryRequestProc : TMemoryRequestProc) : integer; stdcall;
```

Parameters

document
[out] Pointer to a TDocument structure that you should fill with information about the document you want to import.

memoryRequestProc
[in] Pointer to a TMemoryRequestProc callback function that you should call in order to change the *TextData* and *EmbeddedFileData* variables size in order to import text or embedded file.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Return RE_IMPORT_COMPLETED to indicate that there is no document to import and that the import should be ended.

Remarks

MyInfo creates and initializes the *document* structure for you.

See also

TDocument

EndImportSession Function

Use this function to perform cleaning tasks after the import is completed.

Syntax

C/C++:

```
int __stdcall EndImportSession(void);
```

Delphi:

```
function EndImportSession() : integer; stdcall;
```

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

TMemoryRequest Callback Function

Use this function to allocate text or embed file data memory.

Because sharing variable data between DLL and host application is cumbersome, your plugin should call the memory request function, if it needs to import text data or embedded file. Pointer to this function is passed to the plugin at every ImportDocument function call.

Syntax

C/C++:

```
typedef int (__stdcall *TMemoryRequestProc)(TDocument* document, int targetMemory, int newMemorySize);
```

Delphi:

```
type TMemoryRequestProc = function(document : PDocument; targetMemory, newMemorySize : integer) : integer;
stdcall;
```

Parameters

document
[in], [out] Pointer to a TDocument structure that contains a pointer to the data you want to allocate.

targetMemory
[in] Indicates which data you want to request. It should be one of the Target Memory Type constants.

newMemorySize
[in] Indicates the size of the data you request.

Return value

Returns MV_SUCCESS to indicate that memory was allocated successfully.

Return MV_ERROR to indicate that either an error occured, or the request is above MyInfo limits (MyInfo has 4100 KB limit for embedded files).

Example

```
int result = memoryRequestProc(document, MR_TEXT_DATA, 2040);
```

See also

Target Memory Types, ImportDocumentFunction

TPluginInfo Structure

The TPluginInfo structure contains information about the plugin. It is used by the GetPluginInfo function.

Syntax

C/C++:

```
struct TPluginInfo
   {
   char Version[256];
   char PluginName[256];
   char Description[1001];
   char Author[256];
   char Copyright[256];
   int  ImportCaps;
   int  ExportCaps;
   int  ImportTextFormat;
   int  ExportTextFormat;
   char FileExt[51];
   char FileExtDescription[256];
   }
```

Delphi:
see PluginUnit.pas

Members

- **Version**
  Indicates version of the plugin (i.e. "1.23").
- **PluginName**
  Indicates name of the plugin (i.e. "Milenix MyInfo 2.x Import").
- **Description**
  Describes plugin purpose (i.e. "Imports MyInfo 2.x files").
- **Author**
  Indicates plugin author (i.e. "Milenix Software Ltd.").
- **Copyright**
  Contains copyright information (i.e. "Copyright Milenix Software Ltd. 2004").
- **ImportCaps**
  Indicates whether the plugin has import capablities. If so, set ImportCaps to *PI_IMPORT_UTF8* (recommended) or *PI_IMPORT_ANSI* constant. If your plugin has no import capablities, set this variable to *PC_NONE* constant. This variable is set to *PC_NONE* by default.
  - *PI_IMPORT_UTF8* - when this option is used (recommended), MyInfo treats all strings send by the plugin as UTF8 strings.
  - *PI_EXPORT_ANSI* - when this option is used, MyInfo uses system default codepage to translate strings send by the plugin.
- **ExportCaps**
  Indicates whether the plugin has export capablities. If so, set ExportCaps to *PE_EXPORT_UTF8* constant. If your plugin has no export capablities, set this variable to *PC_NONE* constant. This variable is set to *PC_NONE* by default.
- **ImportTextFormat**
  Set it to one of the supported file types (see Text Format Constants for details). MyInfo will treat the *TextData* you import as it is in the type specified here. This variable is set to TD_TEXT by default.
- **ImportTextFormat**
  Set it to one of the supported file types (see Text Format Constants for details). MyInfo will send *TextData* to your plugin in the file format specified here. This variable is set to TD_TEXT by default.
- **FileExt**
  Indicates which file type does the plugin handle (i.e. ".out"). Must begin with dot symbol. Leave empty, if the plugin needs no file open dialog.
- **FileExtDescription**
  Used as description for the file type in the MyInfo dialog (i.e. "MyInfo 2.x Files (*.out)").

See also

GetPluginInfo

TCustomColumn Structure

The TCustomColumn structure contains information about MyInfo tree column.

Syntax

C/C++:

```
struct TCustomColumn
   {
   char Caption[256];
   int  ColType;
   int  Format;
   char Values[5001];
   int  Alignment;
   int  Width;
   int  Visible;
   }
```

Delphi:
see PluginUnit.pas

Members

- **Caption**
  Contains custom column caption (i.e. "Document Approved"). Make sure that the caption is not already used by another column in the topic, otherwise MyInfo will add a number suffix after it's caption and you may be unable to find the column using the GetAvailableCustomColumns function.
- **ColType**
  Indicates custom column type (i.e. CT_YES_NO). The type can be one of the Custom Column Types.
- **Format**
  Indicates custom column format. (i.e. YN_TRUE_FALSE).
- **Values**
  Contains popup list values. It is required only if the custom column is of CT_POPUP_LIST or CT_CATEGORY type. Values should be in the following format: "\n""\n", where is the text of the value and "\n" is new-line character. (i.e. "Approved\nNot Approved\n").
- **Alignment**
  Indicates column alignment (i.e. CA_CENTER). This value can be one of the Custom Column Alignment Constants.
- **Width**
  Indicates column width in pixels.
- **Visible**
  Indicates whether column should be visible. It can be 1 (visible) or 0 (hidden).

See also

ImportCustomColumn, ExportCustomColumn

TDocument Structure

The TDocument structure contains information about MyInfo document.

Syntax

C/C++:

```
struct TDocument
   {
   int  Level;
   char Title[256];
   int  Sensitivity;
   int  Priority;
   char Comment[1025];
   char Tags[1025];
   char Link[256];
   char ColumnValues[10001];
   int  Charset;
   char DateCreated[20];
   char DateModified[20];
   char DateStarted[20];
   char DateDue[20];
   char DateCompleted[20];
   char DateReminder[20];
   int  PercentCompleted;
   int  Id;

   char* TextData;
   int TextDataSize;

   char* EmbeddedFileData;
   int EmbeddedFileDataSize;
   char EmbeddedFileName[256];
   }
```

Delphi:
see PluginUnit.pas

Members

- **Level**
  Null-based index of the document level (i.e. root document will be level 0, its subdocuments will have level 1, their subdocuments will have level 2 and so on).
- **Title**
  Contains document title (i.e. "Approve yearly budget").
- **Sensitivity**
  Indicates document sensitivity. This value can be one of the Document Sensitivity Constants.
- **Priority**
  Indicates document priority. This value can be one of the Document Priority Constants.
- **Comment**
  Contains short comment about tha task (i.e. "Check if this is already completed?").
- **Tags**
  Contains document tags. Each tag is separated by a space (i.e. "phone call MaryAnn").
- **Link**
  Link to an Internet address (i.e. "http://www.milenix.com;").
- **Column Values**
  Contains custom column values for the document (i.e. "John\nAnn\n"). Each values is on it's own line (separated by newline "\n"). If some of the custom columns is of type CT_CATEGORY and the document has more than one value in this column, separate these values with " , " delimiter (i.e. "For review , For approvement").
  Column values are ordered according to the order of calling ImportCustomColumn function calls on import or ExportCustomColumn function calls on export.
- **Charset**
  Indicates the charset of the imported data.
- **DateCreated**
  Date/time when the document was originally created.
- **DateModified**
  Date/time when the document was last modified. Ignored on import.
- **DateStarted**
  Date/time when the document was started as a task.
- **DateDue**
  Date/time when the document is due as a task.
- **DateCompleted**
  Date/time when the document was complated as a task.
- **DateReminder**
  Date/time of the next reminder for this document due date.
- **Id**
  Document Id (i.e. "234"). Ignored on import.

- **TextData**
  Row text data in the file format specifed in *ImportTextFormat* value in TPluginInfo structure.
- **TextDataSize**
  The size of *TextData* variable in bytes.
- **EmbeddedFileData**
  Row data of the file you want to embed in this document.
- **EmbeddedFileDataSize**
  The size of *EmbeddedFileData* variable.
- **EmbeddedFileName**
  The name of the file you want to embed to the document including extension (i.e. "2002 Sales report.xls")

See also

ImportDocument, ExportDocument

Text Format Types

Indicates the text file, in which you will get text data or in which you will pass text data to MyInfo.

Constants

- **TD_TEXT**
  Plain text.
- **TD_RTF**
  Rich Text Format text.
- **TD_RVF**
  RichView text format (see http://www.trichview.com for more information).

See also

TPluginInfo

Custom Column Alignment Constants

Indicates the alignment of the custom column.

Constants

- **CA_LEFT**
  Column is left-aligned.
- **CA_CENTER**
  Column is center-aligned.
- **CA_RIGHT**
  Column is right-aligned (usually used for columns with number values).

See also

TCustomColumn, ImportCustomColumn, ExportCustomColumn

Custom Column Types

Indicates the type of the column.

Constants

- **CT_TEXT**
  Column has text values.
- **CT_NUMBER**
  Column has either integer or fractional number values (depends on column *Format*).
- **CT_CURRENCY**
  Column has currency values.
- **CT_YES_NO**
  Column has yes/no, true/false, on/off or checkbox value (depends on column *Format*).
- **CT_POPUP_LIST**
  Column has text value (possible text values are stored in column *Values* parameter).
- **CT_CATEGORY**
  Column has text value (possible text values are stored in column *Values* parameter).
- **CT_DATE_TIME**
  Column has date/time values.

See also

TCustomColumn, ImportCustomColumn, ExportCustomColumn

Number Column Formats

Indicates the type of the number values in column of CT_NUMBER type.

Constants

- **NF_INTEGER**
  Column has integer numbers (i.e. 12).
- **NF_FRACTIONAL**
  Column has fractional numbers (i.e. 12.51).

See also

TCustomColumn, ImportCustomColumn, ExportCustomColumn

Yes/No Column Formats

Indicates the type of the values in column of CT_YES_NO type.

Constants

- **YF_YES_NO**
  Column has "Yes" and "No" values.
- **YF_TRUE_FALSE**
  Column has "True" and "False".
- **YF_ON_OFF**
  Column has "On" and "Off" values.
- **YF_ICON**
  Column has checkbox icon.

See also

TCustomColumn, ImportCustomColumn, ExportCustomColumn

Document Sensitivity constants

Indicates the sensitivity of the document.

Constants

- **SE_NORMAL**
  Document is public (default).
- **SE_PERSONAL**
  Document is personal.
- **SE_PRIVATE**
  Document is private.
- **SE_CONFIDENTIAL**
  Document is confidential.

See also

TDocument, ImportDocument, ExportDocument

Document Priority constants

Indicates the priority of the document.

Constants

- **PT_LOW**
  Document has low priority.
- **PT_NORMAL**
  Document has normal priority (default).
- **PT_HIGH**
  Document has high priority.

See also

TDocument, ImportDocument, ExportDocument

Target Memory Type constants

Indicates for which data variable in TDocument structure you want to request memory.

Constants

- **MR_TEXT_DATA**
  You want to allocate *TextData* memory.
- **MR_EMBED_FILE_DATA**
  You want to allocate *EmbedFileData* memory.

See also

TMemoryRequest callback function.

Overview

Export plugins are modules used by MyInfo to export documents to third-party file formats. To create export plugin you have to write dynamic-link library (DLL) including several functions, depending on the type of the plugin you want to create.

There are two types of export plugins you can use in MyInfo:

Plugins associated with file format

These plugins are bound to a particular file format, so when the user want to export data using such plugin, MyInfo shows him File Save dialog asking him for the file, where she wants to export data.

To create such plugin you need to write the following functions:

1. GetPluginInfo with the following options in TPluginInfo variable:
   - PE_EXPORT_DATA in *ExportCaps*.
   - Appropriate export text file format in *ExportTextFormat*. See GetPluginInfo for details.
   - File extension of the export files (the same file extension is used for import, if the plugin is combined) in *FileExt*. See GetPluginInfo for details.
   - File filter, which will be used in the File Save dialog in *FileExtDescription*. See GetPluginInfo for details.
2. ConfigurePlugin [optional]
3. BeginExportSession
4. ExportCustomColumn [optional]
5. ExportDocument
6. EndExportSession

Plugins without associated file format

These plugins either show their own dialog to the user when BeginExportSession function is called or export their data to a common location (for example exporting Internet Explorer Favorites does not require user interaction, because this data is stored in common location for the current user).

Such plugin must include the following the functions:

1. GetPluginInfo with the following options in TPluginInfo variable:
   - PE_EXPORT_DATA in *ImportCaps*.
   - Appropriate export text file format in *ExportTextFormat*. See GetPluginInfo for details.
   - Empty string in *FileExt* (it is empty by default).
   - Empry string in *FileExtDescription* (it is empty by default).
2. ConfigurePlugin [optional]
3. BeginExportSession
4. ExportCustomColumn [optional]
5. ExportDocument
6. EndExportSession

Installation

- For MyInfo 6.10 and newer:
  To install a plugin, just place it in its own folder under the "`[Documents and Settings]\[User Name]\Application Data\Milenix\MyInfo\4\Plugins`" folder or use the installation feature from within MyInfo (*Tools* > *Add-ins* > *Plugins* > *Install*).
- For MyInfo 6.09 and older:
  To install a plugin, just place it in the "`[MyInfo folder]\Configuration\Plugins`" folder.
- If you want to install your plugin using installation program, you can obtain Plugins folder location reading the PluginsDir value in the "`HKEY_LOCAL_MACHINE\SOFTWARE\Milenix\MyInfo\6`" key on the target computer.

Export Process

When exporting data from export plugin, MyInfo calls your plugin functions in the following order, depending on the plugin type:

Plugins associated with file format

1. BeginExportSession is called, so you can open the file (stored in *file* variable, where the user has selected to export their data in the File Save dialog.
2. If your plugin contains ExportCustomColumn function, it is called. Use this function to obtain a list of custom tree columns, which you will use when exporting documents. This function will be called for each custom column in the tree.
3. ExportDocument is called. Use this function to pass the information about the document you want to import in MyInfo. This function will be called for every document, the user wants to export.
4. When there are no more documents to export, EndExportSession is called. Here you should free all data that you have used during export.

Plugins without associated file format

1. BeginExportSession is called. Here if your plugin needs export data location, you should display dialog, where the user can select target data location for export. Because your plugin is not associated with file format, *file* variable will be empty.
2. If your plugin contains ExportCustomColumn function, it is called. Use this function to obtain a list of custom tree columns, which you will use when exporting documents. This function will be called for each custom column in the tree.
3. ExportDocument is called. Use this function to pass the information about the document you want to import in MyInfo. This function will be called for every document, the user wants to export.
4. When there are no more documents to export, EndExportSession is called. Here you should free all data that you have used during export.

Export Custom Columns

MyInfo allows you to create custom columns of different types (text, numbers, date and other). You can use this functionallity when exporting documents from MyInfo too! This is realized in simple and elegant manner, so it will be easy for you to add such functionallity to your export plugin.

All you have to do is to include ExportCustomColumn function in your plugin.

Then the process is the following: MyInfo calls your ExportCustomColumn function for every custom column in the tree and you have to collect this information in your plugin.

Then, each time you export document data in your ExportDocument function, you can access the corresponding custom column values in the TDocument *ColumnValues* variable. They will be placed in the same order, in which MyInfo sent you information about its columns and each value will be separated by new-line character (i.e. "First custom column data\nSecond custom column data\n").

BeginExportSession Function

Prepares export plugin for the export process by allocating data, opening the export file or showing the user dialog, where she selects the location for exporting MyInfo data.

Syntax

C/C++:

```
int __stdcall BeginExportSession(char* fileName, HWND callerWindow, HINSTANCE dllInstance);
```

Delphi:

```
function BeginExportSession(fileName : pchar; callerWindow : THandle; dllInstance : longword) : integer;
stdcall;
```

Parameters

file
[in] Pointer to NULL-terminated string containing the full file path and name to the file where you must export the selected data (this varaible is empty string, if your plugin is not associated with file format; in this case, you have to provide the user with dialog where she could select export data location or no dialog at all, if your plugin exports data to a common location).

callerWindow
[in] Use this window handle to make your select location dialog child of the caller dialog.

dllInstance
[in] You may need this DLL handle when creating your select location dialogs.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

ExportCustomColumn Function

Provides information about a custom column in the tree.

Syntax

C/C++:

```
int __stdcall ExportCustomColumn(TCustomColumn* column);
```

Delphi:

```
function ExportCustomColumn(column: PCustomColumn) : integer; stdcall;
```

Parameters

column
[out] Pointer to a TCustomColumn structure that contains information about the custom column you have to export.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Remarks

MyInfo creates and initializes *column* structure for you.

See also

TCustomColumn

ExportDocument Function

Provides information about the document you have to export.

Syntax

C/C++:

```
int __stdcall ExportDocument(TDocument* document);
```

Delphi:

```
function ExportDocument(document: PDocument) : integer; stdcall;
```

Parameters

document
[out] Pointer to a TDocument structure that contains information about the document you have to export.

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

Remarks

MyInfo creates and initializes the *document* structure for you.

See also

TDocument

EndExportSession Function

Use this function to perform cleaning tasks after the export is completed.

Syntax

C/C++:

```
int __stdcall EndExportSession(void);
```

Delphi:

```
function EndExportSession() : integer; stdcall;
```

Return value

Return RE_SUCCESS to indicate success.

Return RE_ERROR to indicate that an error occured.

```
#ifndef PluginHeaderH
#define PluginHeaderH

/* return values */
const int RE_SUCCESS =              1;
const int RE_IMPORT_COMPLETED =     2;
const int RE_ERROR =                3;
const int RE_USER_CANCELED =        4;

/* plugin capabilities */
const int PC_NO_CAPS =              0;

/* import capabilities */
const int PI_IMPORT_DATA =          1;

/* export capabilities */
const int PE_EXPORT_DATA =          1;

/* text data types */
const int TD_TEXT =                 0;
const int TD_RTF =                  1;
const int TD_RVF =                  2;

/* Sensitivity values */
const int SE_NORMAL =               0;
const int SE_PERSONAL =             1;
const int SE_PRIVATE =              2;
const int SE_CONFIDENTIAL =         3;

/* Priority values */
const int PT_LOW =                  0;
const int PT_NORMAL =               1;
const int PT_HIGH =                 2;

/* Custom column types */
const int CT_TEXT =                 0;
const int CT_NUMBER =               1;
const int CT_CURRENCY =             2;
const int CT_YES_NO =               3;
const int CT_POPUP_LIST =           4;
const int CT_CATEGORY =             5;
const int CT_DATE_TIME =            6;

/* CT_NUMBER formats */
const int NF_INTEGER =              0;
const int NF_FRACTIONAL =           1;

/* CT_YES_NO formats */
const int YF_YES_NO =               0;
const int YF_TRUE_FALSE =           1;
const int YF_ON_OFF =               2;
const int YF_ICON =                 3;

/* Custom column alignments */
const int CA_LEFT_AL =              0;
const int CA_RIGHT_AL =             1;
const int CA_CENTER_AL =            2;

/* Memory request options */
const int MR_TEXT_DATA =            1;
const int MR_EMBEDDED_FILE_DATA =   2;

/* Memory request return values */
const int MV_ERROR =                0;
const int MV_SUCCESS =              1;

typedef struct
    {
    char Version[256];
    char PluginName[256];
    char Description[1001];
    char Author[256];
    char Copyright[256];
    int ImportCaps;
    int ExportCaps;
    int ImportTextFormat;
```

```
    int ExportTextFormat;
    char FileExt[51];
    char FileExtDescription[256];
    } TPluginInfo;

typedef struct
    {
    char Caption[256];
    int Type;
    int Format;
    char Values[5001];
    int Alignment;
    int Width;
    int Visible;
    } TCustomColumn;

typedef struct
    {
    int Level;
    char Title[256];
    int Sensitivity;
    int Priority;
    char Comment[1025];
    char Tags[1025];
    char Link[256];
    char ColumnValues[10001];
    int Charset;
    char DateCreated[20];
    char DateModified[20];
    char DateStarted[20];
    char DateDue[20];
    char DateCompleted[20];
    char DateReminder[20];
    int PercentCompleted;
    int Id; // ignored on import

    char* TextData;
    int TextDataSize;

    char EmbeddedFileName[256];
    char* EmbeddedFileData;
    int EmbeddedFileDataSize;
    } TDocument;

typedef int (__stdcall *TMemoryRequestProc)(TDocument* document, int targetMemory, int newMemorySize);

#endif
```

```
unit PluginUnit;

interface
const
    { return values }
    RE_SUCCESS = 1;
    RE_IMPORT_COMPLETED = 2;
    RE_ERROR = 3;
    RE_USER_CANCELED = 4;

    { plugin capabilities }
    PC_NO_CAPS = 0;

    { import capabilities }
    PI_IMPORT_DATA = 1;

    { export capabilities }
    PE_EXPORT_DATA = 1;

    { text data types }
    TD_TEXT = 0;
    TD_RTF = 1;
    TD_RVF = 2;

    { Sensitivity values }
    SE_NORMAL = 0;
    SE_PERSONAL = 1;
    SE_PRIVATE = 2;
    SE_CONFIDENTIAL = 3;

    { Priority values }
    PT_LOW = 0;
    PT_NORMAL = 1;
    PT_HIGH = 2;

    { Custom column types }
    CT_TEXT = 0;
    CT_NUMBER = 1;
    CT_CURRENCY = 2;
    CT_YES_NO = 3;
    CT_POPUP_LIST = 4;
    CT_CATEGORY = 5;
    CT_DATE_TIME = 6;

    { CT_NUMBER formats }
    NF_INTEGER = 0;
    NF_FRACTIONAL = 1;

    { CT_YES_NO formats }
    YF_YES_NO = 0;
    YF_TRUE_FALSE = 1;
    YF_ON_OFF = 2;
    YF_ICON = 3;

    { Custom column alignments }
    CA_LEFT = 0;
    CA_RIGHT = 1;
    CA_CENTER = 2;

    { Memory request options }
    MR_TEXT_DATA = 1;
    MR_EMBEDDED_FILE_DATA = 2;

    { Memory request return values }
    MV_ERROR = 0;
    MV_SUCCESS = 1;

type TPluginInfo = record
    Version,
    PluginName : array[0..255] of char;
    Description : array[0..1000] of char;
    Author,
    Copyright : array[0..255] of char;
    ImportCaps,
    ExportCaps,
    ImportTextFormat,
    ExportTextFormat : integer;
    FileExt : array[0..50] of char;
    FileExtDescription : array[0..255] of char;
    end;
```

```
    PPluginInfo = ^TPluginInfo;

type TCustomColumn = record
    Caption : array[0..255] of char;
    ColType,
    Format : integer;
    Values : array[0..5000] of char;
    Alignment,
    Width,
    Visible : integer;
    end;

    PCustomColumn = ^TCustomColumn;

type TDocument = record
    Level : integer;
    Title : array[0..255] of char;
    Sensitivity : integer;
    Priority : integer;
    Comment : array[0..1024] of char;
    Tags : array[0..1024] of char;
    Link : array[0..255] of char;
    ColumnValues : array[0..10000] of char;
    Charset : integer;
    DateCreated : array[0..19] of char;
    DateModified : array[0..19] of char;
    DateStarted : array[0..19] of char;
    DateDue : array[0..19] of char;
    DateCompleted : array[0..19] of char;
    DateReminder : array[0..19] of char;
    PercentCompleted : integer;
    Id : integer; // ignored on import

    TextData : pchar;
    TextDataSize : integer;

    EmbeddedFileName : array[0..255] of char;
    EmbeddedFileData : pchar;
    EmbeddedFileDataSize : integer;
    end;

    PDocument = ^TDocument;

type
    TMemoryRequestProc = function(document : PDocument; targetMemory, newMemorySize : integer) : integer; stdcall;

implementation
begin

end.
```

Plugin Icon

Starting with MyInfo 6.10, your plugin can have its own icon in MyInfo plugin dialogs.

There are two ways to include icon for your plugin:

- **(Recommended)** Add small icon (16x16) as resource with name "IDI_SMALL_ICON" and large icon (32x32) as resource with name "IDI_LARGE_ICON" to your plugin DLL file.
- Or ship .ico file, combining both 16x16 and 32x32 icons, named as your DLL file (e.g. if your plugin DLL name is MyPlugin.dll, your icon should be named MyPlugin.ico).

If your plugin has no icon, MyInfo will use generic one. Including icon to the plugin does not prevent it from working with older versions of MyInfo.

Send To MyInfo API

If you are a developer of a Windows application, you can add capabilities to send information directly to MyInfo as a new document.

You need two things:

1. Get a handle to the MyInfo application window
2. Send the information

Getting handle

In order to obtain handle to the MyInfo application window, you should use the FindWindow Windows API function and pass "MyInfoMain4" as a class name parameter. If the function fails the first time, maybe MyInfo is not started. You can start it using ShellExecute Windows API function passing the application executable path found in the "AppExe" string value in the "SOFTWARE\\Milenix\\MyInfo\\4" key of Windows Registry.

If FindWindow fails again, then there is probably some problem with MyInfo installation and you should notify the user.

Here is an example using Visual C++:

```
// find app window
HWND hWnd = FindWindow("MyInfoMain4", "");
// open app if no window is found
if(0 == hWnd)
{
  CComBSTR appExe;
  appExe = GetStringValue("SOFTWARE\\Milenix\\MyInfo\\4", "AppExe");

  if(StrCmp(OLE2T(appExe), "") != 0)
    {
      ShellExecute (0, "open", OLE2T(appExe), 0, 0, SW_SHOWNORMAL);
      Sleep(2000);

      hWnd = FindWindow("MyInfoMain4", 0);
    }
}

// if app could not be open, exit
if(0 == hWnd)
{
  MessageBox(0, "Could not open Milenix MyInfo!\nPlease check if MyInfo is installed properly.", "Send to MyInfo", MB_ICONINFORMATION);
  return;
}
```

Sending the information

In order to send a message to MyInfo, you should use the WM_COPYDATA Windows API message.

MyInfo support 5 commands:

Value	Description
1	Activates MyInfo application window. No information is sent.
2	Launches myinfo:// link to MyInfo file or document.
3	Imports web document (creates a new document with link to a web page).
4	Imports text document (creates a new document and inserts text data in it).
5	Imports HTML document (creates a new document and inserts HTML data in it).

Command format

Each command is sent in plain text (or in UTF8 for the HTML content) and has the following format:

```
[MyInfo identificator][separator][command][separator][command specific options]
```

where [MyInfo identificator] should be "MyInfo Command", [separator] is a special character with ANSI number of 1 (or \x01 hex), the [command] is some of the commands described above, and [command specific options] are fields with additional information, depending on the command you send.

According to the command, these are the additional fields (all separated by the separator character descibed above:

Command	Fields	Example (in C++)
1 - Active	No additional fields.	-
2 - Launch link	Link in the myinfo:// protocol format.	`MyInfoCommand\0x011\0x01myinfo://c:\\myinfo.mio`
3 - Import link	Link to an Internet address and document title.	`MyInfoCommand\0x012\0x01http://www.milenix.com\0x01Milenix`
4 - Send text	Text (ANSI or RTF) and document title.	`MyInfoCommand\0x013\0x01Sample text\0x01Sample title`
5 - Send HTML	HTML fragment, document title and base import URL (optional: non-utf8 text indicator = 1).	`MyInfoCommand\0x014\0x01Sample HTML with <a href="link">link</a>\0x01Sample title\0x01http://www.milenix.com`

Here is an example using Visual C++:

```
std::string sendString;
std::string separator = "\x01";
sendString.append("MyInfoCommand");
sendString.append(separator);
sendString.append("5");
sendString.append(separator);
sendString.append(OLE2T(textSelection));
sendString.append(separator);
sendString.append(OLE2T(title));
sendString.append(separator);
sendString.append(OLE2T(link));
sendString.append(separator);
sendString.append("1"); // tell MyInfo that this text is not UTF8
sendString.append(separator);

COPYDATASTRUCT copyData;
copyData.dwData = 0;
copyData.cbData = sendString.length() + 1;
copyData.lpData = (void*)sendString.c_str();

SendMessage(hWnd, WM_COPYDATA, 0, (long)&copyData);
```