
# GP Power Tools for
Microsoft Dynamics™ GP

## User's Guide

## Build 31

Application & documentation designed, developed, and supported by

**David Musgrave of Winthrop Development Consultants**

# Contents

# Chapter 1: Introduction

GP Power Tools is a primarily Dexterity based with Visual C# and Visual Basic.Net components suite of utilities and tools created to assist with the task of supporting Microsoft Dynamics GP.

GP Power Tools is divided into four modules which can be purchased separately with some standard features available to all modules. The modules are:

- Administrator Tools
- Developer Tools
- Form Control Tools
- Database Tools

The System Module which is automatically registered when one or more of the three modules above is registered.

The features of GP Power Tools are also divided into User level (Standard Mode) and Administrator level (Advanced Mode) features.

Standard Mode features are read-only and can be safely used by all users in a system. Advanced Mode features include scripting and accessing system settings and should only be used by system Administrators. To access an Advanced Mode feature, a user will need elevated privileges at the SQL Server level in addition to application-level security and an optional system or administrator password.

On a registered system, if you open a window from an unregistered module, you might be asked if you wish to open the window in Preview Mode. If a window is opened in Preview Mode, you may use the window to explore its features, however its functionality will be limited.

Below is a list of what features are contained in each module with Advanced Mode features highlighted with an asterisk (*).

The **System Module** contains the following features:

| Feature | Description |
|---|---|
| Logging Control including Manual Logging Mode | Manually turn on SQL Logging and Dexterity Logging and Profiling |
| ScreenShot | Capture and either email or save Screenshots and System Status information |
| Send Email | Send Email messages from within the application |
| Calculator | Touch friendly standard calculator with clipboard integration |
| Dex.ini Settings | Change System and Debugger Dex.ini Settings for the current workstation |
| Administrator Password Setup* | Create optional separate password to be used when accessing Advanced mode features |
| Logging Settings* | Change system wide Logging Settings such as shared path location, default logs and SQL Profile Trace setup |
| Email Settings* | Change system wide Email Settings controlling the email engine used by the tool |
| Configuration Export/Import* | Export and Import settings |
| Configuration Maintenance* | Clear GP Power Tools data tables |
| Setup Backup and Restore* | Backup all data in SQL Tables to Debugger.xml file and restore from Debugger.xml to SQL Tables |
| Dictionary Assembly Generator Control* | Check the status of Dictionary Assembly DLL files and recreate them if necessary |

The **Administrator Tools** module contains the following features:

| Feature | Description |
|---|---|
| Resource Information | Obtain Details of any Table, Form, Window, Field, Report, Script, Global Variable, Constant or Message/Warning resource |
| Resource Finder | Enhanced window to locate table data related to any field in any product |
| Security Profiler | Monitor all Security check activity |
| Security Information | Display Security settings for specific resources for a user and company |
| Security Log | Security activity tracking for users, companies and the entire system |
| Security Analyzer | Displays results of various queries against the Security data for the system to highlight possible security related issues |
| Enhanced Security | The primary interface to Deny Based Security additional security layer including denying security and hiding menu navigation options on a per user per company basis |
| Security Denied | Deny Based Security maintenance window for security denied |
| Security Hidden | Deny Based Security maintenance window for security hidden |
| Administrator Settings* | Change system wide Administrator Settings controlling the behavior of the tool, including Company Color Themes and Automatic Logout |
| Dex.ini Configuration* | Automatically update Dex.ini settings across multiple workstations |
| Dictionary Control* | Enable and disable third party products and VBA and Visual Studio customizations |
| Company Login Filter* | Filter companies available when logging in based on the installation folder and/or launch file name used |
| Window Position Memory* | Automatically remember a user's preferred window position, size and state for any window in Microsoft Dynamics GP |
| User Activity Log* | User Login Activity tracking to record logins and logouts and track daily maximum session count on a system, user and company basis |
| Login Limits* | Limit user logins on a system, user and company basis |
| Launch File Configuration* | Automatically update Dynamics.set launch files across multiple workstations |
| Dynamic Product Selection* | Allows selection between multiple versions of the same window or report as the window or report is opened |
| Website Settings* | Change the website used for the Connect and Intelligent Cloud Insights homepage sections |
| Product Version Validation* | Identify when there are mismatched product dictionaries installed or when product dictionaries are not installed when all products are needed |

The **Developer Tools** module contains the following features:

| Feature | Description |
|---|---|
| Trigger Status | Display currently active triggers |
| Runtime Executer | Run published Dexterity sanScript scripts |
| SQL Executer | Run published Transact SQL scripts |
| .Net Executer | Run published C# and VB.Net scripts |
| Project Setup* | Group together all the related Triggers, Scripts (Dexterity, SQL, .Net) and Parameter Lists into a Project |
| Trigger Setup* | Create triggers for debugging or custom development |
| Runtime Execute Setup* | Create and run Dexterity sanScript scripts |
| SQL Execute Setup* | Create and run Transact SQL scripts |
| .Net Execute Setup* | Create and run C# and VB.Net scripts |
| Snippet Setup* | Create and maintain script snippets |
| Parameter Lists* | Create interactive parameter dialogs to be used with scripting features. |
| Messages Setup* | Create reusable multi-lingual messages for use in triggers and scripts |
| Dynamic Trigger Logging* | Track execution paths by dynamically registering triggers against events in Dynamics GP and logging when they occur |

The **Form Control Tools** module contains the following features:

| Feature | Description |
|---|---|
| Form Control Status* | Display currently active Form Control triggers |
| Form Control Resources* | Display the form resource data for forms with Form Control applied |
| Form Control Setup* | Setup Form Control for no code/low code customization of forms |
| Password Setup* | Setup Passwords for use with Form Control and Developer Tools |

The **Database Tools** module contains the following features:

| Feature | Description |
|---|---|
| XML Table Export* | Export any table(s) to an XML file |
| XML Table Import* | Import previously exported tables |
| Database Validation* | Validate SQL user and database information and table structures |
| SQL Login Maintenance* | Reset Users' Passwords and view or change password policy settings |
| Password Reset Email Settings* | Control settings for sending emails when resetting user passwords |
| Copy User Settings* | Copy user settings in the system database between user IDs |
| SQL Trigger Control* | Disable, enable and delete SQL table triggers for troubleshooting or maintenance |
| Note Fix Utility* | Identify and fix issues with Record Notes and Note Index values |
| Database Space Recovery* | Recovery unused space from database tables and enable/disable compression. |

## Examples of use

GP Power Tools has many uses. Here are some examples:

- When issues or bugs occur, GP Power Tools helps you identify the specific series of events that led up to them.
- Got performance problems? Use GP Power Tools to quickly and simply turn on all logging and profiling capabilities without restarting GP.
- Do you want to know the details about dictionary resources? GP Power Tools gives you a complete and in-depth look at all security objects, including Forms, Windows, Tables, Reports, Fields, and Scripts.
- Find out what's causing security access issues by using GP Power Tools to identify the responsible forms, reports, or tables.
- Deny security access to individual resources on a per user per company basis without needing to duplicate security tasks and roles.
- To help with troubleshooting issues, GP Power Tools can easily enable or disable third party products or change the order of the products in the launch file.
- When you need to import or export data to any GP table at all, think GP Power Tools.
- Do you need to run SQL, Dexterity, C# or VB.Net scripts? You can do it with GP Power Tools even if you don't have Dexterity, Visual Studio or SQL Administration Tools installed.
- Capture, save, and email screenshots of all open windows and send a system status report at the same time with GP Power Tools.
- Even if your local system doesn't have Outlook installed, GP Power Tools can be used to send email to the system administrator.
- Overcome those difficult Report Writer (RW) user-defined function issues with GP Power Tools.
- GP Power Tools makes it easy to roll out Dex.ini setting changes to all workstation in your system.
- Before doing a GP upgrade, GP Power Tools can validate your SQL users, databases and table structures to help ensure it goes smoothly.
- How about creating brand new Security Tasks and Roles by tracing user activity that you capture interactively in GP Power Tools or from security activity tracking logs?
- Need extra functionality for a web service integration, create custom web services with GP Power Tools.

- Want to customize behavior on windows throughout the application without using multiple triggers and scripts, the Form Control module has you covered.

- Want to add fields to windows without needing to use Modifier to create a Modified window, use Virtual Fields.

## Support

Support for GP Power Tools is provided by Winthrop Development Consultants.

Support cases can be logged using the link below:

https://www.winthropdc.com/support.htm

## GP Power Tools Portal

You can also find release histories, FAQ documents and lots of articles as well as links to download and purchase at the GP Power Tools Portal:

http://WinthropDC.com/GPPT

# Chapter 2: Installation and Configuration

This chapter includes the following sections:

- *Installation*
- *Security*
- *Navigation*
- *Recommended Configuration*
- *SQL Profile Tracing Configuration*
- *Macro Recording Configuration*
- *About GP Power Tools*
- *GP Power Tools Registration*
- *GP Power Tools Update Check*
- *GP Power Tools Feedback Survey*
- *Advanced Mode Access*
- *GP Power Tools and the Web Client*

## Installation

GP Power Tools is installed by downloading the installer and executing it. Follow the onscreen instructions to install the product files into the Microsoft Dynamics GP application folder and the Addins subfolder. GP Power Tools must be installed on all workstations and servers to be fully functional.

The installation contains the following files:

- GPPTools.cnk (self-installing dictionary)
- GPPTools.txt (readme file)
- GPPTools.pdf (this user guide manual)
- Documentation/GPPTools.pdf (copy of this user guide manual)
- GPPTools_License.doc (the license agreement which you accept by using the tool)
- Dex.chm (Dexterity Help file)
- DAG.EXE (Dictionary Assembly Generator tool)
- Application.GpPowerTools.dll (signed Dictionary Assembly)
- Application.GpPowerTools.xml (IntelliSense data for Visual Studio)
- Application.GpPowerTools.Metadata.dll (signed Dictionary Assembly for Service Enabled Procedures)
- Application.GpPowerTools.Metadata.xml (IntelliSense data for Visual Studio)
- Addins/WinthropDC.GpPowerToolsVC.dll (Visual C# support)
- Addins/WinthropDC.GpPowerToolsVB.dll (Visual Basic.Net support)
- MimeKit.dll (For mail support to Office 365 added in Build 29)
- MailKit.dll (For mail support to Office 365 added in Build 29)
- System.Buffers.dll (For mail support to Office 365 added in Build 29)
- System.Memory.dll (For mail support to Office 365 added in Build 29)
- System.Runtime.CompilerServices.Unsafe.dll (For mail support to Office 365 added in Build 29)

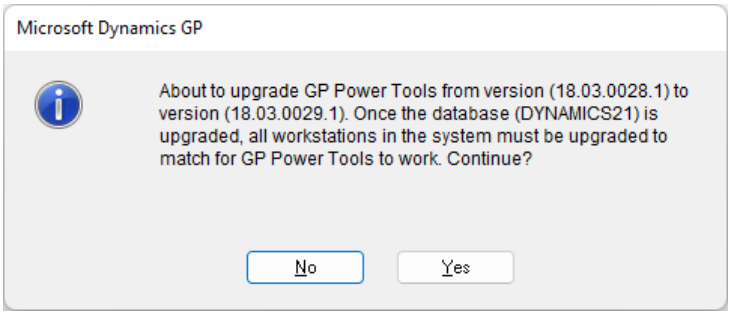
Check the properties of all the dll files and Unblock them if necessary.

When Microsoft Dynamics GP is next launched, if asked, select “Yes” to include new code.

*If installing on a Windows system with User Account Control (UAC) active, please launch Microsoft Dynamics GP with the Run as Administrator option to complete the installation.*

Log into Microsoft Dynamics GP with a user the appropriate SQL privileges to create tables, such as 'sa' or 'DYNSA'. GP Power Tools will automatically create its SQL tables in the system database.

If a previous build of GP Power Tools is already installed in the system, a dialog to confirm you want to update the tables will be displayed.




*Once the tables are updated to a new version on a system, the installations of GP Power Tools on every workstation and server must be updated to match the newly installed build.*

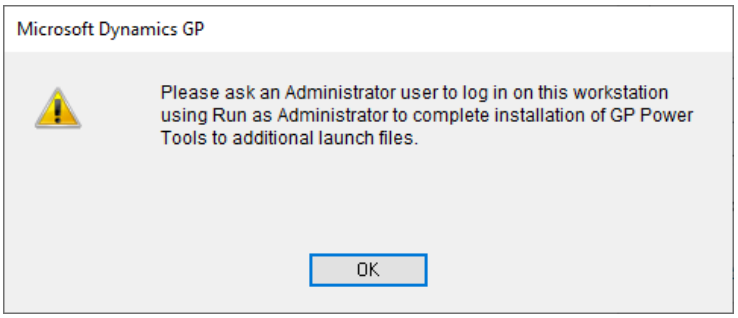
If you had a previous Support Debugging Tool installation, GP Power Tools will read the Debugger.xml settings file to populate the initial data in the SQL tables.

*From Build 29 onwards, the installation will automatically apply the Windows Bitmap Scaling settings to fix display issues when the display is set to more than 100% DPI. The settings can be checked or changed on the Dex.ini Settings window.*

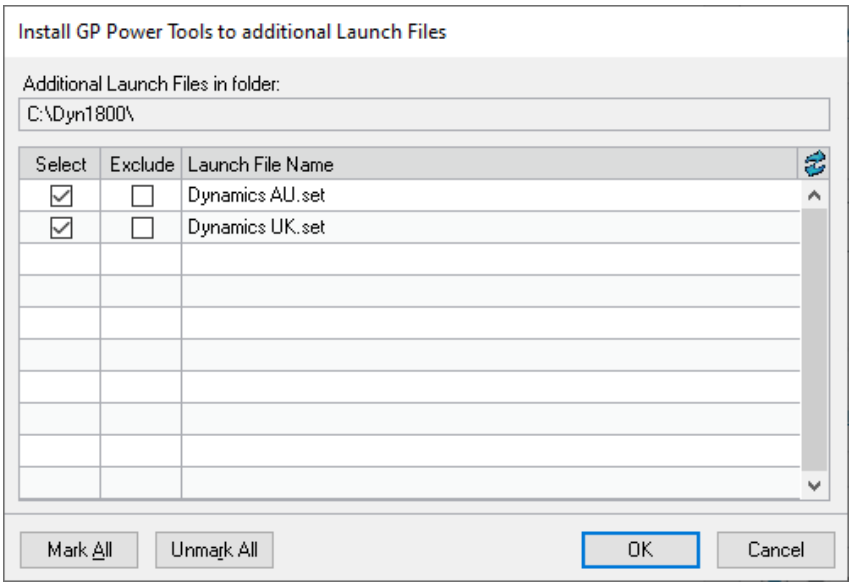
## Additional Launch File Installer

If GP Power Tools is installed on workstation which has additional launch files in the application folder, it will offer to install itself into the additional launch files.

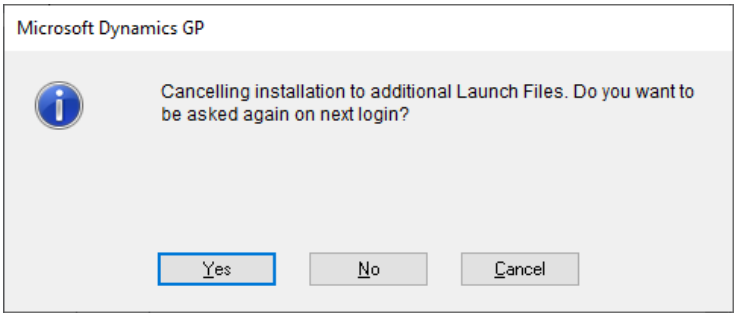
If a non-Administrator user logs in when there are additional launch files waiting to be updated, the following message will be displayed. Login will proceed as normal once the message is dismissed.




When an Administrator user logs in and there are additional launch files waiting to be updated, the following dialog will be displayed. Select the launch files you want updated to add GP Power Tools or mark them as excluded. Excluded launch files will not be included in the dialog for future installs.



Clicking OK will update the selected launch files and record that the installation has been completed. If Cancel is clicked, no changes are made, and the following dialog will confirm if the installation should be recorded as completed.




*The MBS_Debug_Install Dex.ini setting is used to track if the installation on the current workstation has been completed. The WDC_InstallExclude Dex.ini setting is used to track the excluded launch file names (separated by commas).*

## Security

Security access must be granted to the forms of GP Power Tools before it can be used by users other that those belonging to the POWERUSER security role.

GP Power Tools will automatically create the Security Tasks and Security Roles required to use the tool. The following Security Roles are created.

GP POWER TOOLS USER (GP Power Tools User)

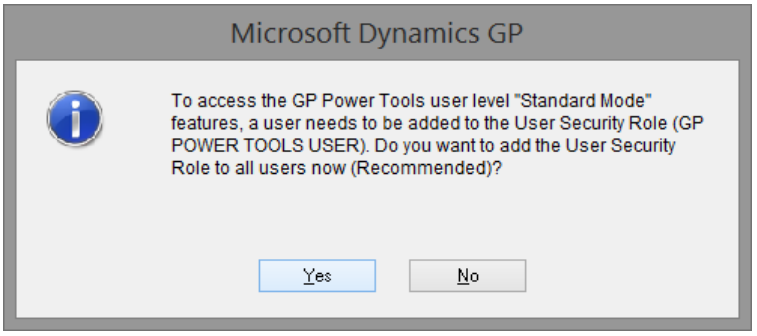
GP POWER TOOLS ADMIN (GP Power Tools Administrator)

GP POWER TOOLS PASSWORD (GP Power Tools Administrator Password)

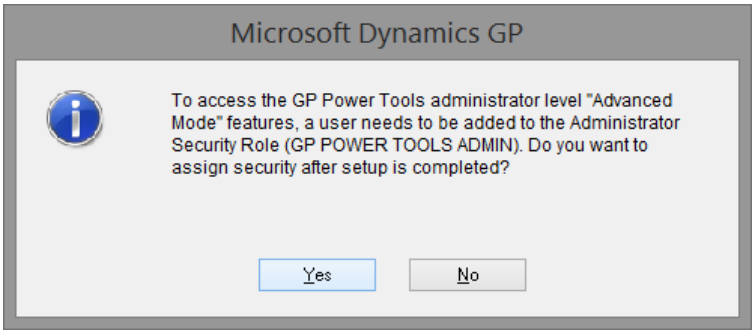
GP POWER TOOLS SERVICES (GP Power Tools Services) for GP 2015 or later.

The administrator security role grants access to all areas of the tool, while the user security role only grants access to the Standard Mode features. Advanced Mode features are only available to Microsoft Dynamics GP User IDs that also have the SQL Server sysadmin fixed server role or membership of the db_owner role on the system database (DYNAMICS) and the current company database, even if security is granted.

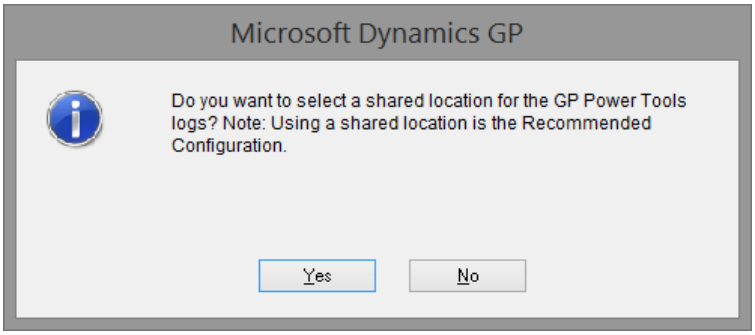
After installing GP Power Tools: If logging into Microsoft Dynamics GP as a user belonging to the POWERUSER security role, and no users have been granted access to the GP POWER TOOLS USER security role, the system will offer to add this security role to all users for you.



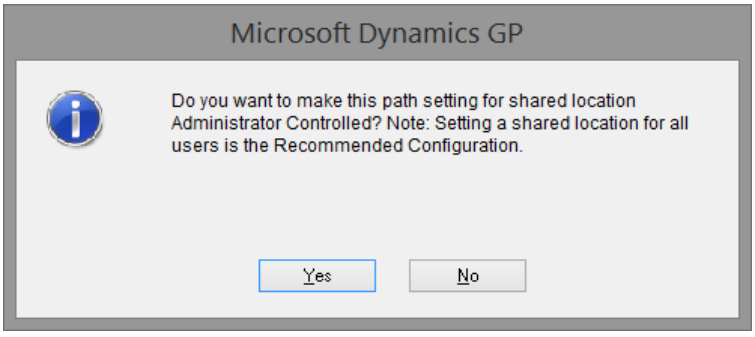

If you respond Yes, the system will remind to you to add the GP POWER TOOLS ADMIN security role to other users who need access to the Advanced Mode features (and do not already have access to the POWERUSER Security Role). You have the option to open the User Security Setup window when setup is completed.




You will then have the option to select a shared location for logs and export files to be stored in. If you select No, the default location is the Data folder in the application folder for Microsoft Dynamics GP. If you select Yes, you will be presented with a dialog to select the path you wish to use. This path should point to a folder which has full control permissions for all users and can be specified using either a UNC pathname or a shared drive letter available to all users.




If you selected a shared location, then you will be asked if you would like to make this Administrator Controlled. Making the setting Administrator Controlled, automatically rolls the setting out to all workstations in the system on their next login and is the Recommended Configuration.



To manually grant security to the forms of GP Power Tools use the User Security Setup window (Microsoft Dynamics GP >> Tools >> Setup >> System >> User Security). After selecting the user and company, select one of the security roles below:

GP POWER TOOLS USER (GP Power Tools User)

GP POWER TOOLS ADMIN (GP Power Tools Administrator)

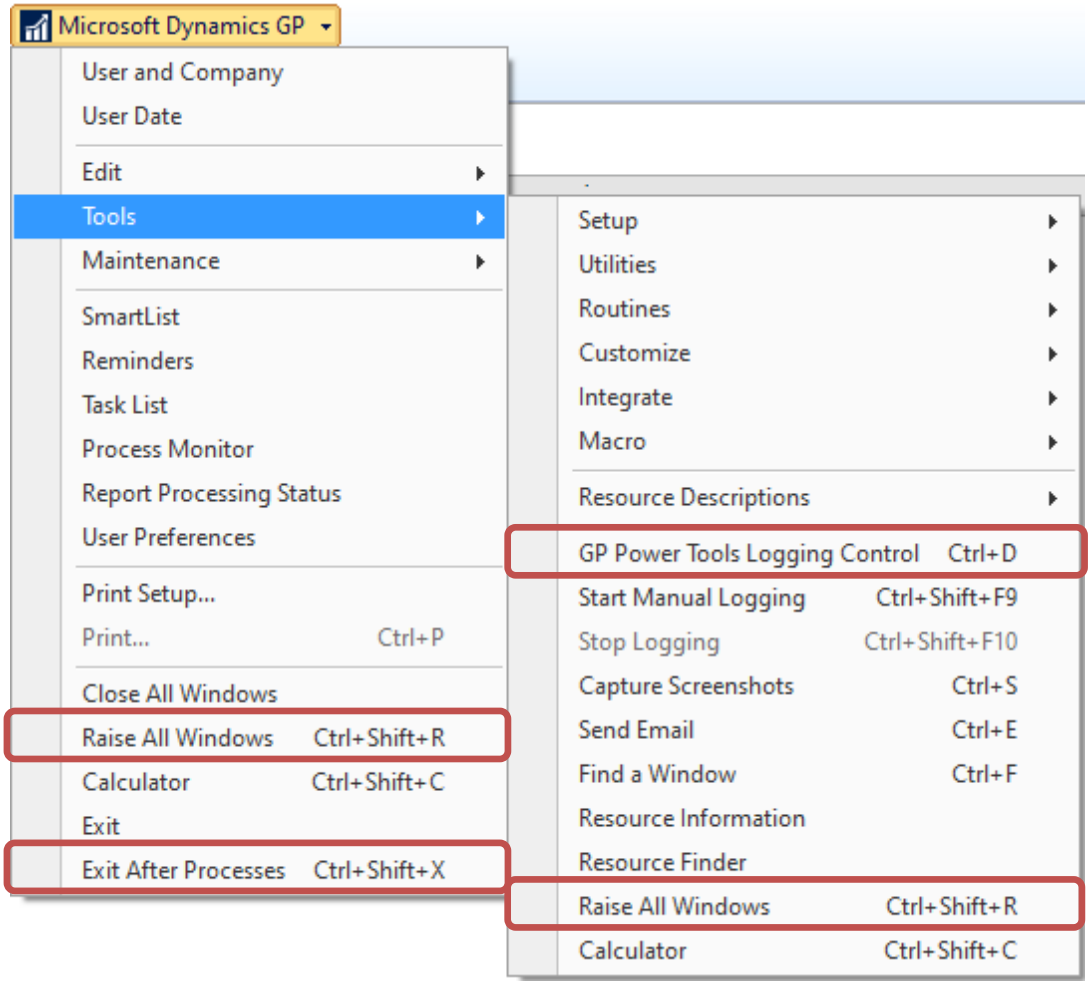
GP POWER TOOLS PASSWORD (GP Power Tools Administrator Password)

GP POWER TOOLS SERVICES (GP Power Tools Services) for GP 2015 or later.

*If a user is not going to be using any of the windows of GP Power Tools, they do not need to be assigned to a security role. Automatic Trigger Mode will work regardless of security settings.*

## Navigation

Once logged into Microsoft Dynamics GP, a user with security access granted can find GP Power Tools Logging Control main window under the Tools menu underneath the Microsoft Dynamics GP menu (highlighted below). It also has the keyboard shortcut Ctrl+D assigned to it.

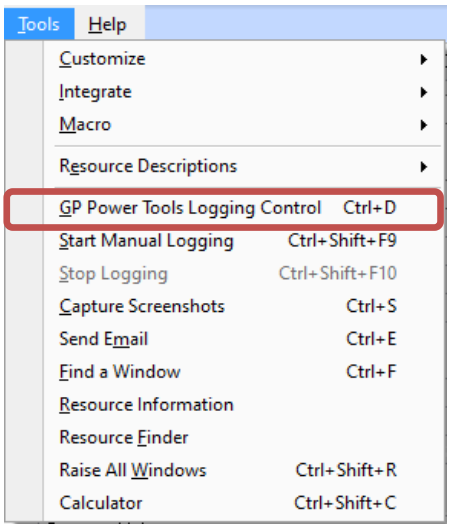



*GP Power Tools also adds the Raise All Windows option to the main application menu and Tools menus, to allow for an easy method to send the main application window to the background. The option has the keyboard shortcut Ctrl+Shift+R assigned to it.*

*Also, added to the main application menu is the Exit After Processes option, which will request the application to exit after it has completed all background processing. The option has the keyboard shortcut Ctrl+Shift+X assigned to it.*

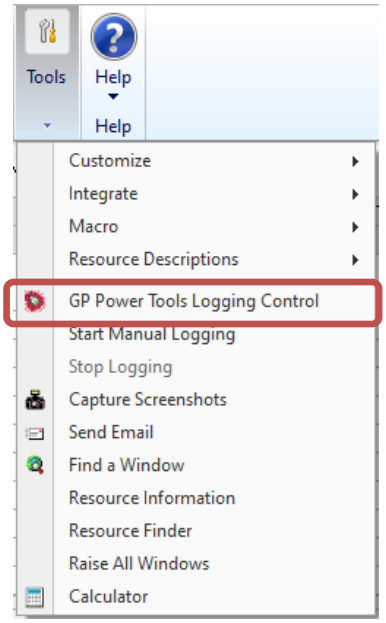
From the GP Power Tools Logging Control main window, the Options button drop list can be used to open other windows.

In addition, GP Power Tools is also found under the Tools menu on each individual window of Microsoft Dynamics GP (highlighted below).

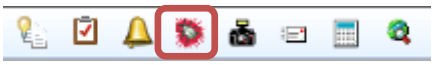



You may need to press and release the Alt key on the keyboard to allow the window menu bar to activate before the shortcut keys work.

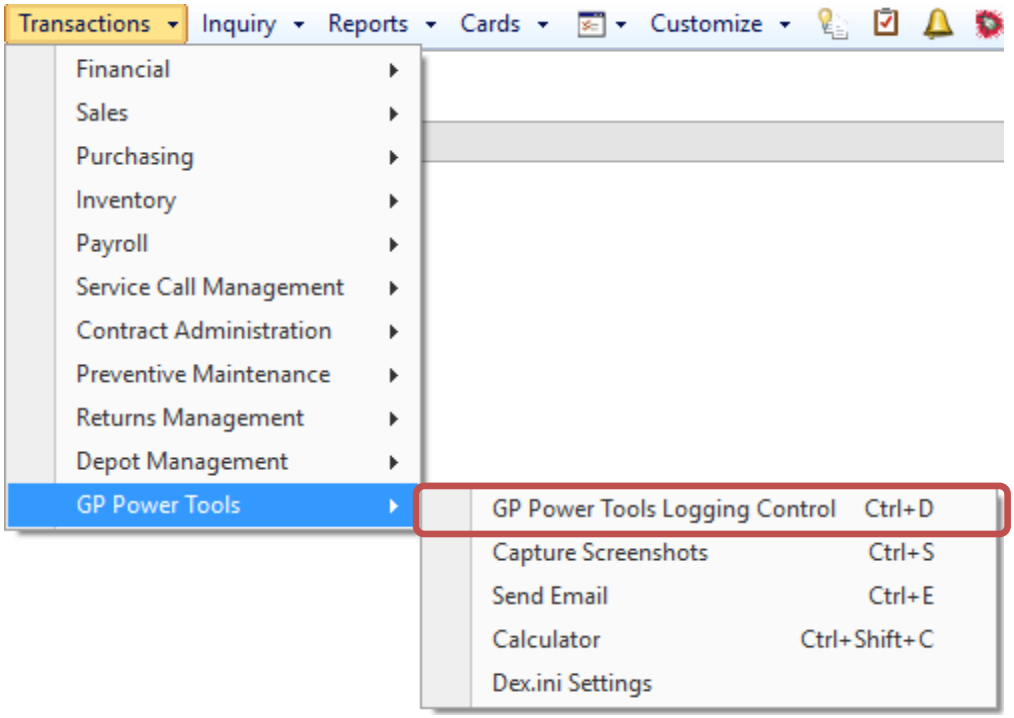
If using Microsoft Dynamics GP 2013 R2 or later in desktop mode with ribbons enabled instead of the menus, you can access GP Power Tools under the Tools button on the ribbon.




GP Power Tools can also be opened from the Standard Toolbar and from Quick Links on the Home Page.



All GP Power Tools windows are also available via the standard application menus under the GP Power Tools submenus. The GP Power Tools Logging Control main window can be found under Transactions >> GP Power Tools >> GP Power Tools Logging Control.

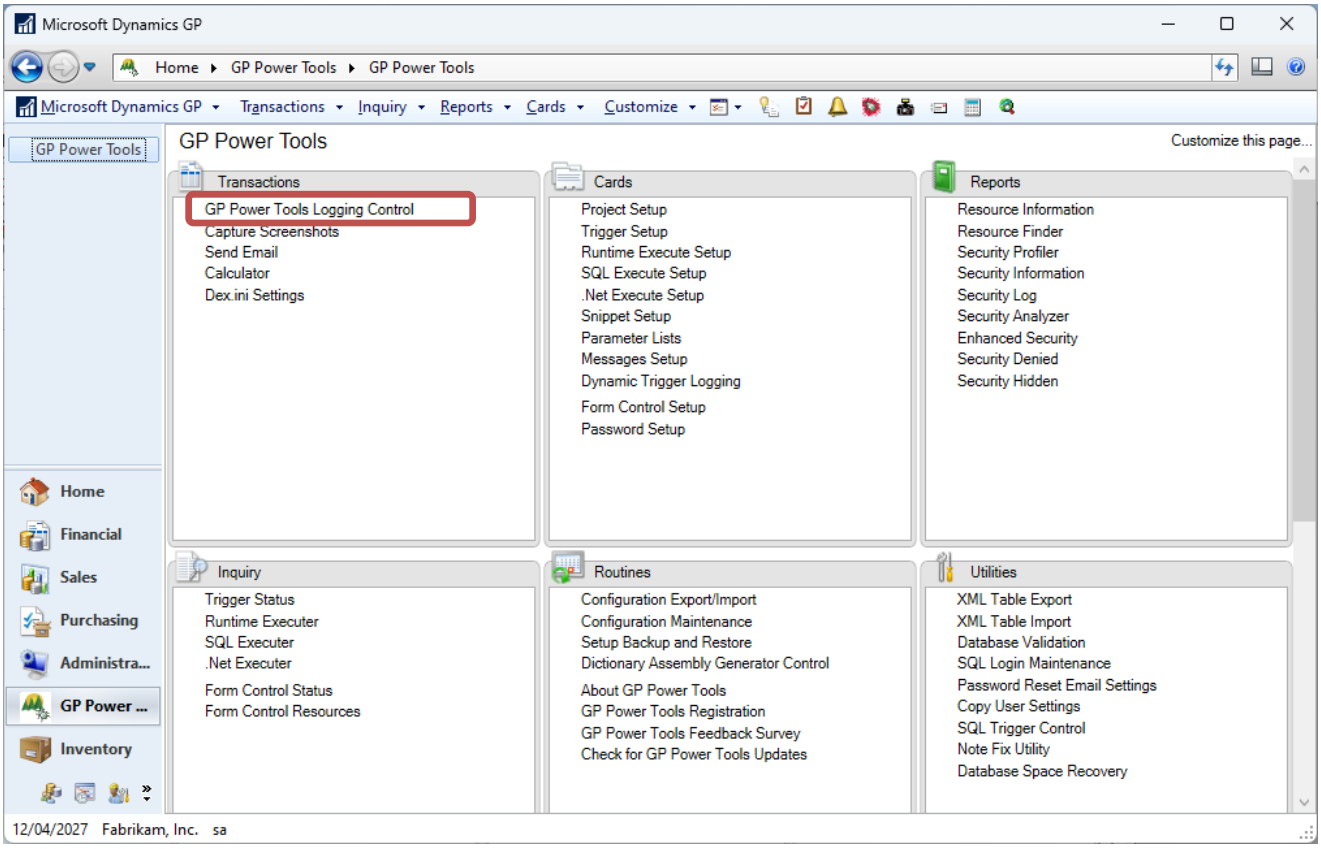



Finally, you can use the GP Power Tools Area Page by clicking on the GP Power Tools Navigation Pane button.

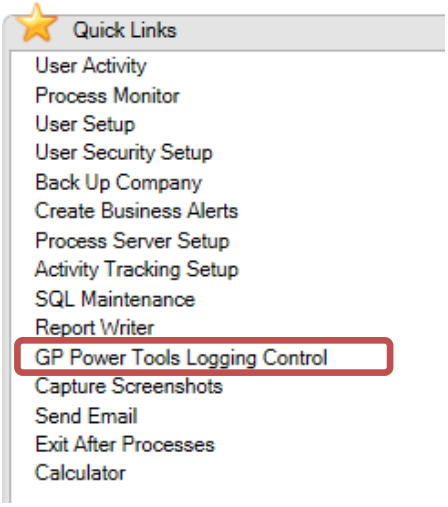

*If the GP Power Tools button or icon are not visible, you might need to use the Navigation Pane Options or Show More Buttons menus from the bottom of the Navigation Pane.*

*GP Power Tools adds the Find a Window option to the main application menu and window level Tools menu. This opens a normally hidden Microsoft Dynamics GP core window that can search the navigation menus for matching items and open them for you. The option has the keyboard shortcut Ctrl+F assigned to it. The Find a Window icon is also added to the Standard Toolbar.*

Once the GP Power Tools Area Page is displayed, all the various windows will be displayed, including the main GP Power Tools Logging Control window (under Transactions).




When running on the Web Client, use the GP Power Tools area page or the Quick Links on the Home Page to open GP Power Tools.



## Recommended Configuration

GP Power Tools stores its settings in SQL tables. If the SQL tables are empty and you had a previous install, the XML setup file called Debugger.xml while be accessed once to load the data into the SQL tables.

While a shared location is no longer required for the data storage it is recommended for the storage of all logs and export files created. This avoids having to visit an individual workstation to have access to the files.

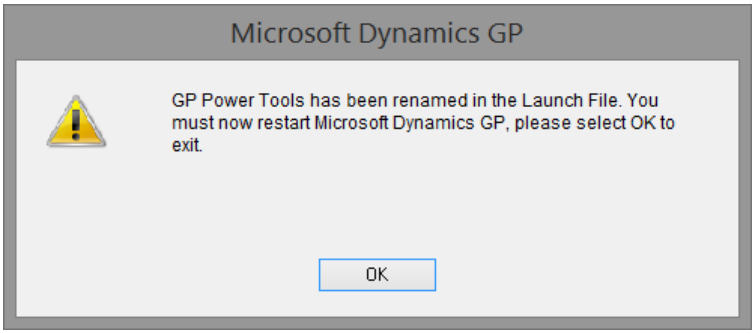
The recommended configuration is for GP Power Tools to be installed on all workstations in the system and to point each workstation to use a single shared location.

Below are step by step instructions to install and set up the Recommended Configuration:

1. Initially install on a single instance of Microsoft Dynamics GP.

2. Launch Microsoft Dynamics GP using Run as Administrator and click Yes if asked "Do you wish to include new code now?"

3. If upgrading from a previous install, you might be asked to re-launch Microsoft Dynamics GP after changes were made to the Dynamics.set launch file. If, so go back to step 2.

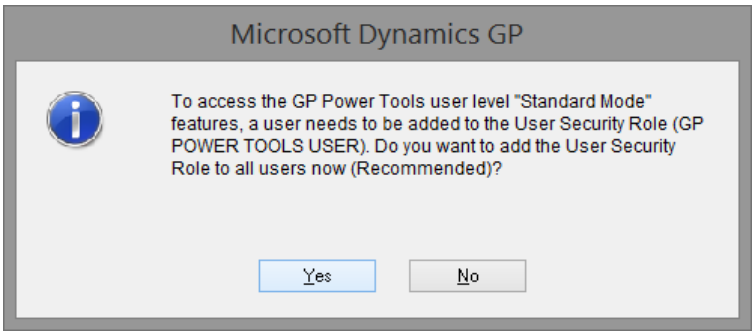



4. Log into Microsoft Dynamics as 'sa' or a user with similar permissions.

5. If the SQL tables need to be created or updated, you will see a progress dialog in the bottom right of your screen as the tables are created. Any existing data will be preserved.

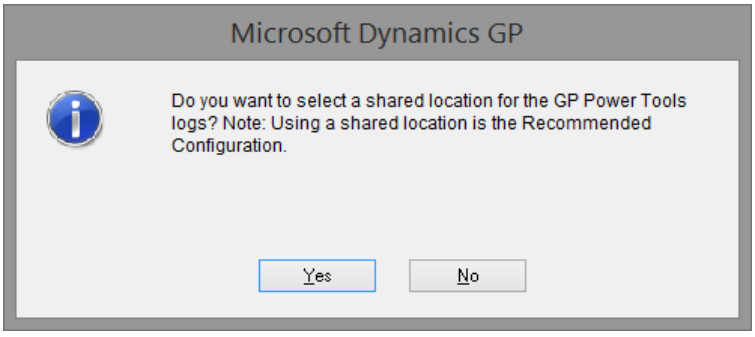
6. If upgrading a previous install, the SQL tables are empty and a Debugger.xml file can be located, it will be read to populate the SQL tables. You can import a different Debugger.xml file later if desired.

7. If asked to add the base user level of security access to all users, click Yes.

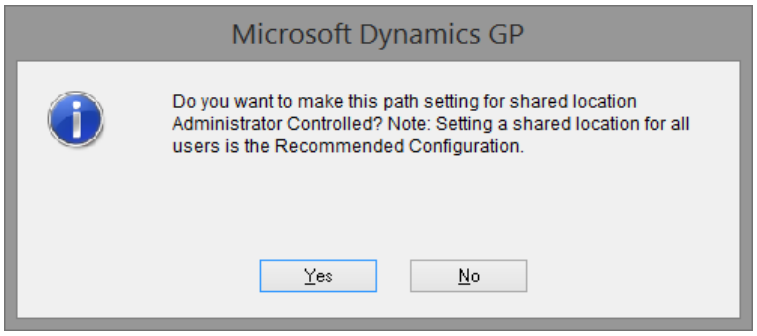



8. You will then be reminded that administrator level security settings will need to be set up manually. You can ask to open the User Security Setup window when setup is completed.


9. If asked to select a shared location for the setup files and logs, click Yes and select the path you wish to use. This path should point to a folder which has full control permissions for all users and can be specified using either a UNC pathname or a shared drive letter available to all users.



10. If asked about making the path setting for the shared location Administrator Controlled, click Yes.




11. **Optional:** To manually change security settings, go to the User Security Setup window (Administration >> Setup >> System >> User Security), select the appropriate user and company and grant access to one or both of the following roles:

    For user features:
    GP POWER TOOLS USER (GP Power Tools User)
    GP POWER TOOLS SERVICES (GP Power Tools Services) for GP 2015 or later.

    For administrator features:
    GP POWER TOOLS ADMIN (GP Power Tools Administrator)
    GP POWER TOOLS PASSWORD (GP Power Tools Administrator Password)

*It is recommended to grant all users in the system access to GP POWER TOOLS USER. Only System Administrators need access to GP POWER TOOLS ADMIN unless they already have access to the POWERUSER Security Role.*

12. Install GP Power Tools on all other workstations in the system.

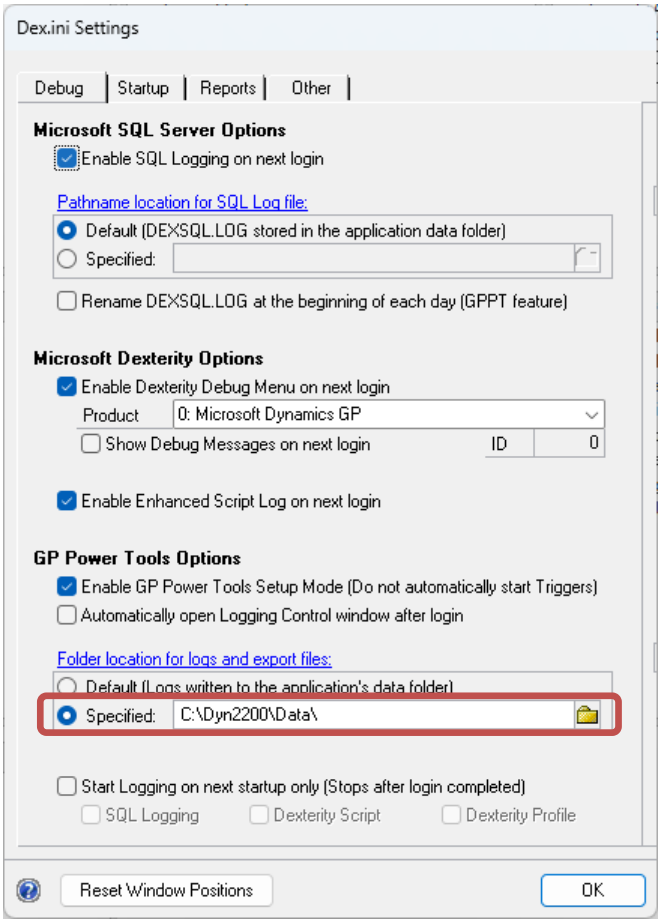
The Recommended Configuration is now configured. To install on other workstations just requires the copying of the files and the including of new code.

Below are the manual steps showing where the responses to the dialogs for Folder Location settings can be manually changed:

To update the current workstation only:

1. Open the Dex.ini Settings window by selecting Dex.ini Settings from the Cards section of the GP Power Tools Area Page or by selecting Dex.ini Settings from the Options button drop list on the main window.

2. From the Dex.ini Settings window, on the Debug tab, select a Specified Pathname location for logs and export files.



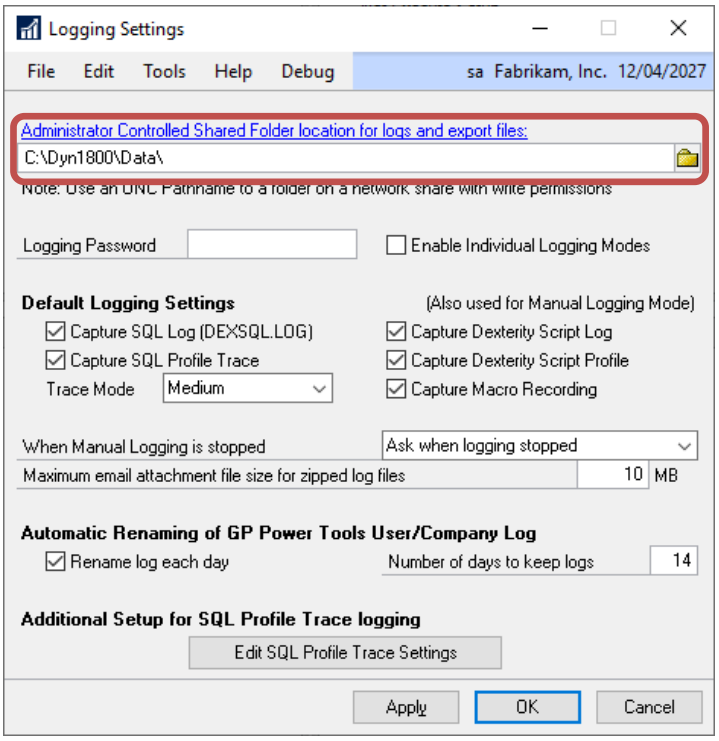

*The pathname can be specified using a UNC path in the format \\Server\Share\Folder\.*

3. Click OK to save the changes.

To update the Administrator controlled setting:

1. Open the Logging Settings window by selecting Logging Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Logging Settings from the Options button drop list on the main window.

2. From the Logging Settings window, select a shared folder where all logs and export files will be written. This path should point to a folder which has full control permissions for all users and can be specified using either a UNC pathname or a shared drive letter available to all users.




*The pathname can be specified using a UNC path in the format \\Server\Share\Folder\.*

3. Click OK to save the changes.

That is all that is required for the Recommended Configuration.

## SQL Profile Tracing Configuration

For more information on setting up and enabling SQL Profile Tracing please see the section under the Logging Settings window.

Below are step by step instructions to configure the recommended settings for SQL Profile Tracing:

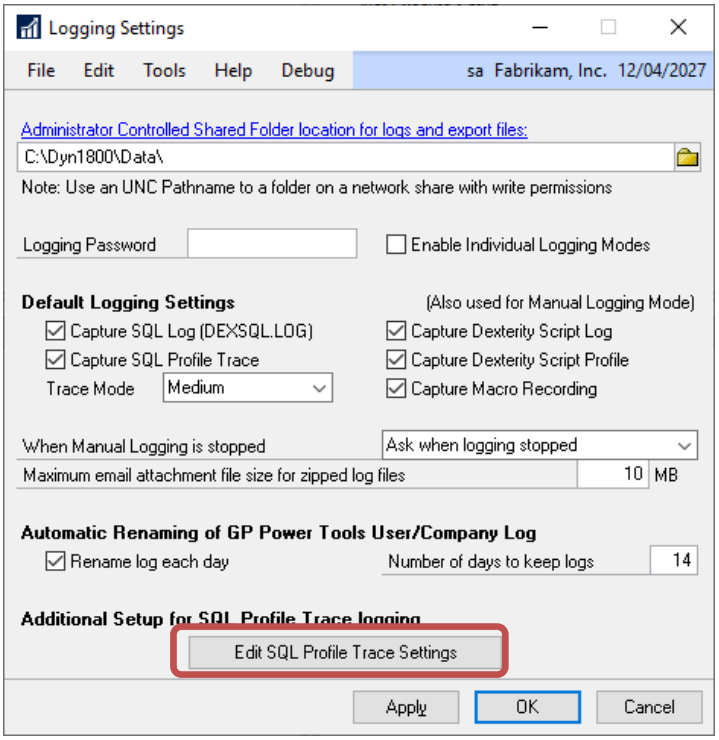
1. On the SQL Server machine create a folder on a local drive for where the SQL Profile Trace files will be stored while they are being created. Note this local path for later.

2. Share this local folder on the network, so that all Microsoft Dynamics GP users will have Full Control to the folder. Note this network UNC path for later.

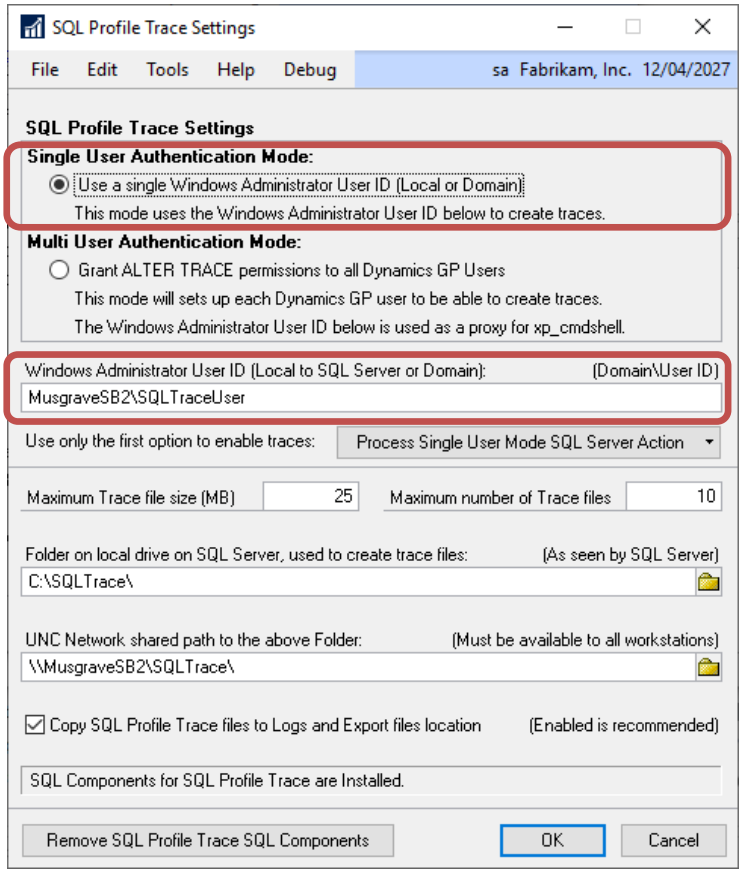
3. Create a user (for example: SQLTraceUser) to be used by SQL Profile Tracing system. The user can be a local user on the SQL Server or a domain user, but needs local Administrator rights on the SQL Server machine. It is recommended to set the password to not expire. Note the User ID and password for later.

4. Log into Microsoft Dynamics as 'sa' or a user with similar permissions. Open the Logging Settings window by selecting Logging Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Logging Settings from the Options button drop list on the main window.

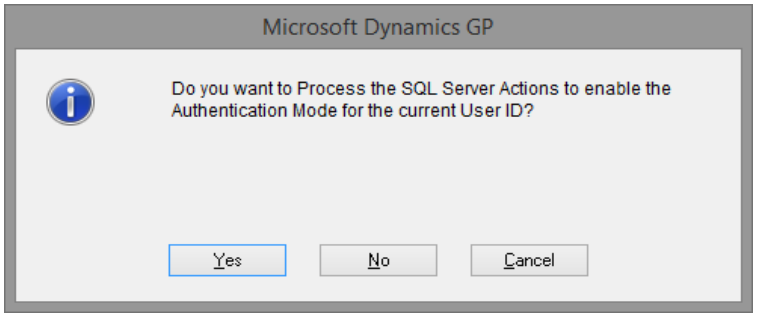
5. From the Logging Settings window, click Edit SQL Profile Trace Settings to open the SQL Profile Trace Settings window.



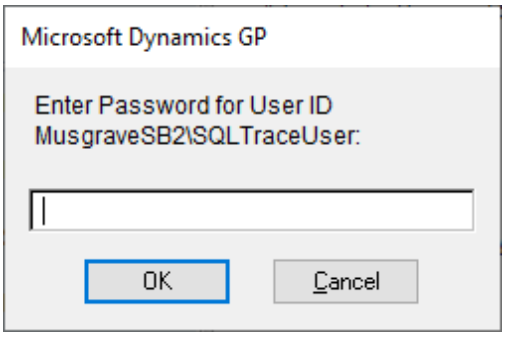
6. On the SQL Profile Trace Settings window, make sure Single User Authentication Mode is selected. In this mode only the single user created earlier will need permissions to create SQL Traces and the permissions for individual users do not need to be changed or elevated.




7. Enter the user created previously and press tab. The system will then ask if you want to process the SQL Server Actions to enable the Authentication Mode, click Yes.

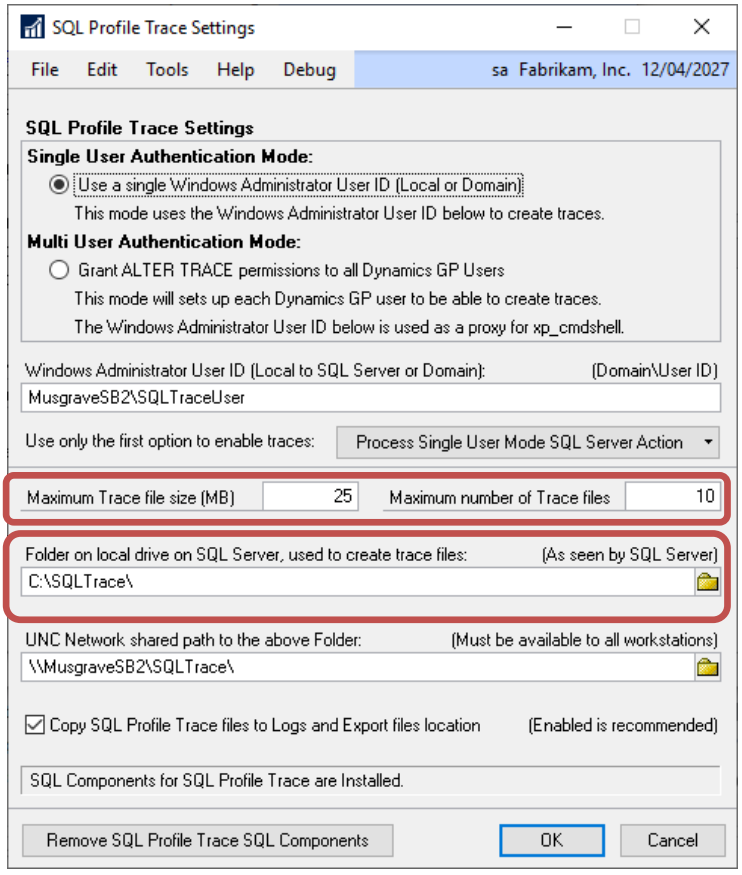


8. As each step of the SQL Server actions needed to enable the Authentication Mode are completed a desktop alert will be displayed. You will also be asked for the password for the user for the Enable xp_cmdshell proxy account step. The password is not validated at this time, so please ensure it is entered correctly.

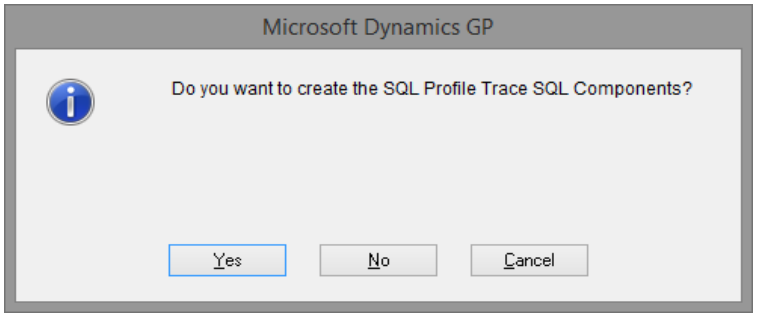



*To see the list of individual steps for enabling or disabling the Authentication Mode, click the Process Single User Mode SQL Server Action or Process Multi User Mode SQL Server Action button. You can select to manually run all of the steps or select individual steps from the list.*

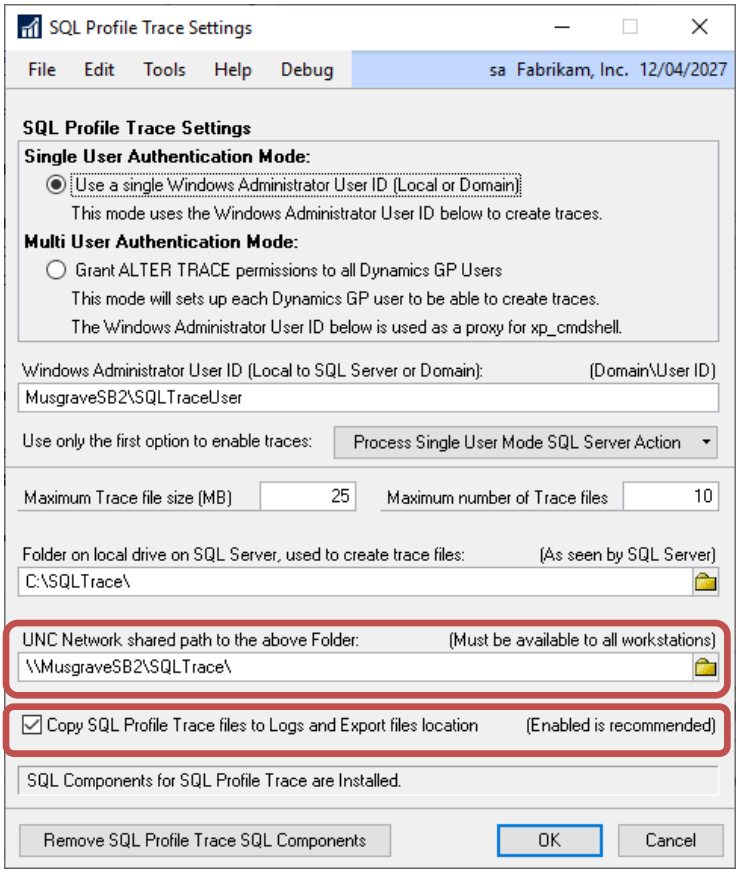
9. You can change the Maximum Trace file size and Maximum number of Trace files if desired, or just leave the default values.



10. Enter in the Local Path set up previously (as created in step 1) and press tab. The system will then ask if you want to create the SQL Profile Trace SQL Components, click Yes to create the stored Procedures in the DYNAMICS system database.



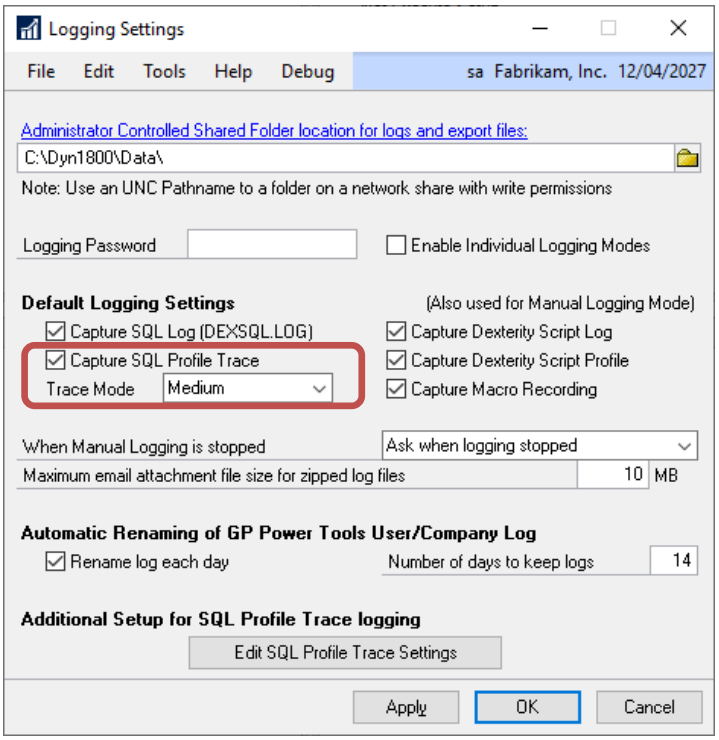

11. Enter the UNC Network Path set up previously (as created in step 2) and press tab.




12. Make sure the Copy SQL Profile Trace files to Debugger Settings location option is enabled. This will copy the completed trace files from the SQL Server to the folder used for the Debugger Settings and logs.

13. Click OK to save the settings and close the SQL Profile Trace window.

14. On the Logging Settings window, enable the Capture SQL Profile Trace option and set the desired Trace Mode (use Small, if unsure). This will enable SQL Profile Tracing for Manual Logging Mode and as the default value for Trigger Setup.




15. Click OK to save the settings and close the Logging Settings window.

## Macro Recording Configuration

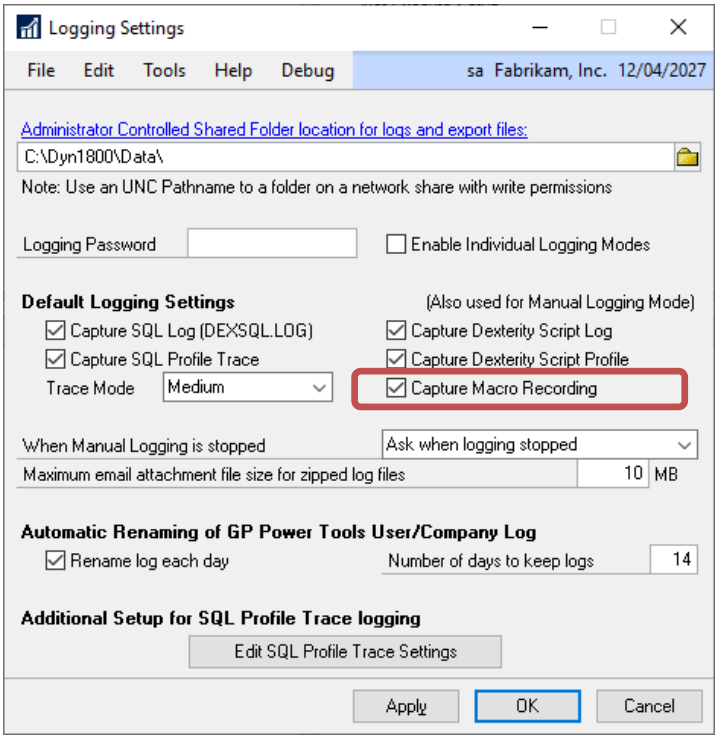
For more information on enabling Macro Recording please see the section under the Logging Settings window.

Below are step by step instructions to enable Macro Recording:

1. Log into Microsoft Dynamics as 'sa' or a user with similar permissions.

2. Open the Logging Settings window by selecting Logging Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Logging Settings from the Options button drop list on the main window.

3. On the Logging Settings window, enable the Capture Macro Recording option. This will enable Macro Recording for Manual Logging Mode and as the default value for Trigger Setup.

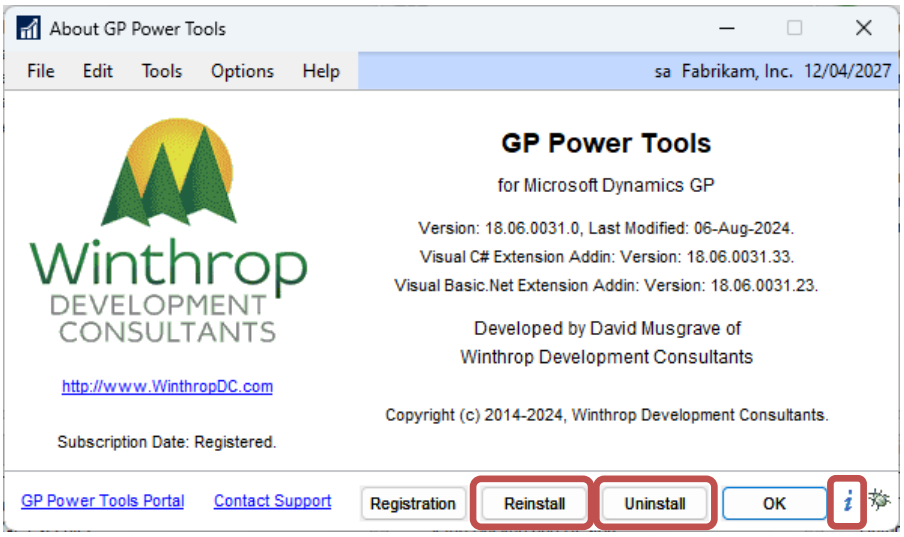



4. Click OK to save the settings and close the Logging Settings window.

## About GP Power Tools

You can open the About GP Power Tools window by selecting About GP Power Tools from the Routines section of the GP Power Tools Area Page or by selecting About GP Power Tools from the Options button drop list on the main window.

The About GP Power Tools window shows the current version, build and last modified date information.

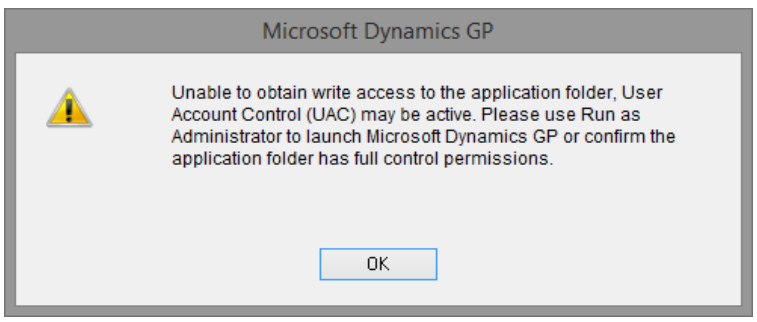



You can uninstall GP Power Tools from this window. Clicking Uninstall will remove GP Power Tools from the menus and security tables and remove any Dex.ini settings added.

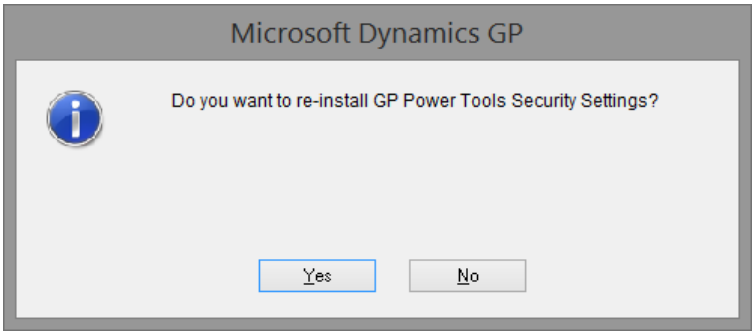
If SQL Profile Tracing is enabled, you will be asked if you want to remove the SQL Server permissions and components created by GP Power Tools.

You will also be asked if you want the Dynamics.set launch file updated to remove GP Power Tools, so that it does not re-install itself next time Microsoft Dynamics GP is launched.

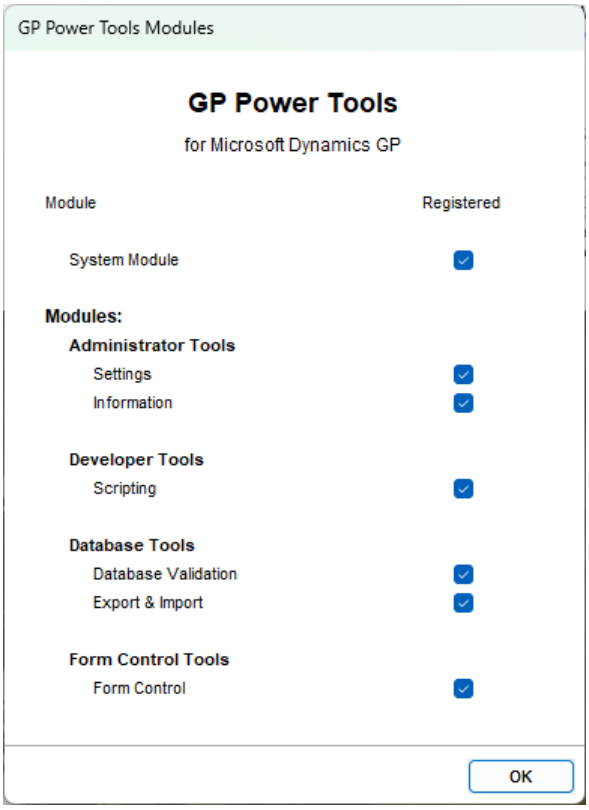
*If User Account Control (UAC) is preventing writer access to the application folder, you will see the following dialog displayed. You will need to use Run as Administrator to allow access and complete the un-install.*



You can also re-install GP Power Tools from this window. Clicking Re-install will, after a confirmation dialog, remove GP Power Tools from the menus and security tables, then re-run the installation as discussed in the Installation section above.




To check which modules are registered you can click on the Info button.

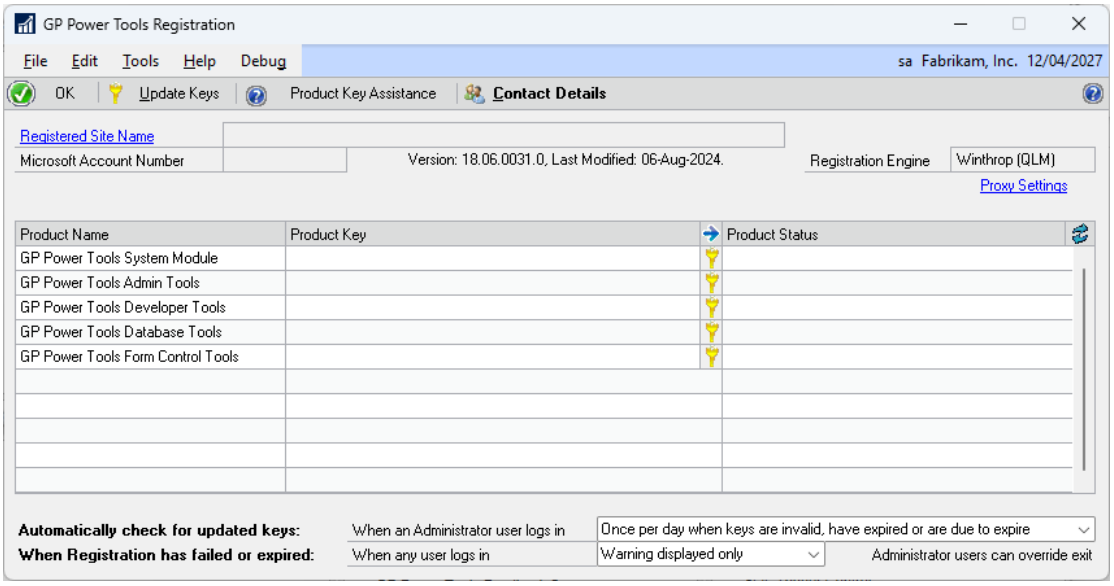


## GP Power Tools Registration

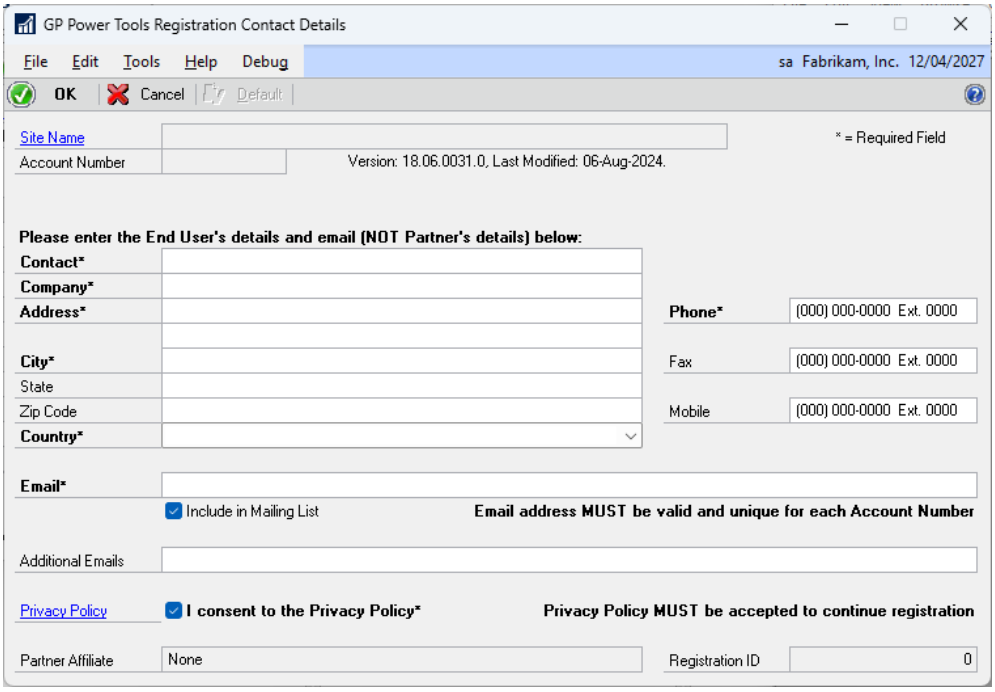
You can open the GP Power Tools Registration window by selecting GP Power Tools Registration from the Routines section of the GP Power Tools Area Page or by selecting GP Power Tools Registration from the Options button drop list on the main window.

The GP Power Tools Registration window can also be opened by clicking the Registration Button on the About GP Power Tools window, or from the Additional menu on the Microsoft Dynamics GP Registration window.

The current Microsoft Dynamics GP registration details of the system along with the registration keys for each module will be displayed.




Use the Contact Details button to complete or update the Contact Details for the site, including consenting to the Privacy Policy. The Privacy Policy must be accepted for the OK Button to be enabled.




*The contact details must be for the end user of the system and not for a partner. The Email address must be the valid address for the end user as it must be unique. The Email address links the site details with the registration keys.*

Use the Update Keys button to retrieve current keys for the system from the registration server.

Use the individual Trial Key button to retrieve the key for that module and if no key is available, request a 30 day trial of that module.

If a Product Key has been manually provided, it entered directly into the Product Key field on the window. It is recommended to use copy and paste to minimize typing errors.

The Automatically check for updated keys option can be used to control the frequency that the system will automatically request updated keys from the registration server when the current keys are expired or optionally due to expire.

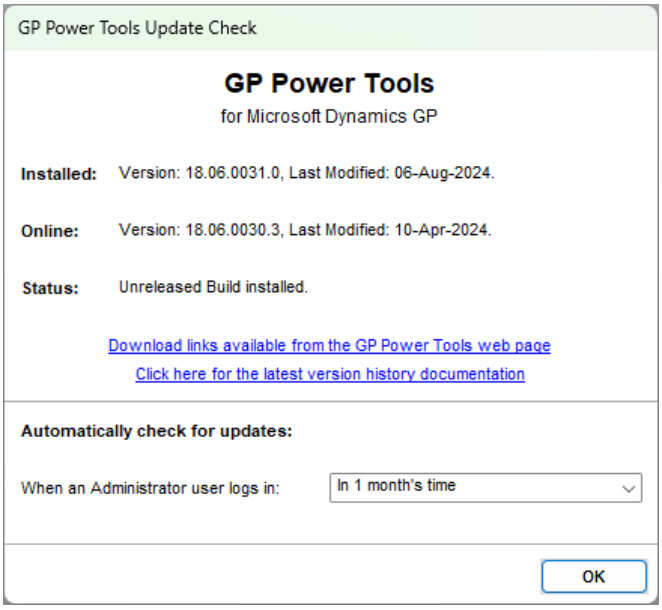
*The Automatic check for updated keys only is executed when an Administrator user logs in.*

When a previous Registration has failed or expired, the system can be configured to optionally warn users on login or prevent them from accessing the system. This is especially useful when there are settings or customizations using GP Power Tools which must always be active on the system.

## GP Power Tools Update Check

GP Power Tools can automatically check online to see if an updated build or hotfix has been released.

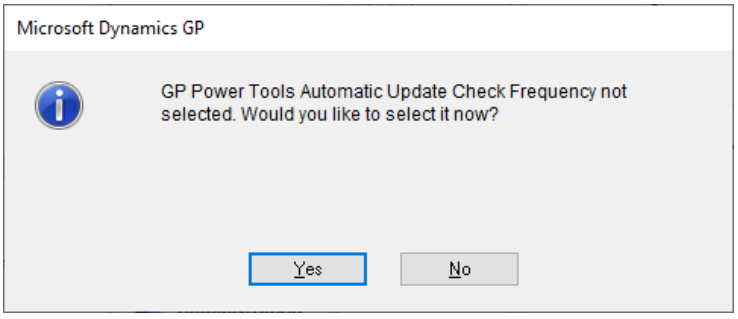
The GP Power Tools Update Check window can be opened by selecting Check for GP Power Tools Updates from the Routines section of the GP Power Tools Area Page or by selecting Check for GP Power Tools Updates from the Options button drop list on the main GP Power Tools Logging Control window. It can also be opened by using the Options menu on the About GP Power Tools window and selecting Check for Updates.




The Automatically check for updates option can be used to control the frequency that the system will check for updates.

*If an update is available, you can select not to be notified again for this update. This will skip the one update and notify you when the next update is available.*

If the frequency has not been set, the following dialog will be displayed. Select Yes to open the GP Power Tools Update Check window.

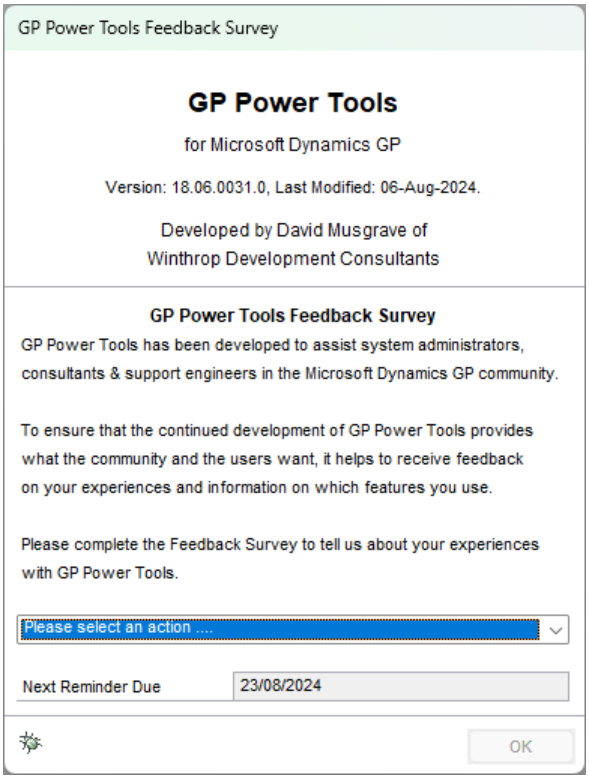



*The Automatic check for update only is executed when an Administrator user logs in. This check is for information only, no updates to the system will be made.*

## GP Power Tools Feedback Survey

GP Power Tools includes a dialog to prompt users to provide feedback via an online survey (http://WinthropDC.com/GPPT/Survey.htm).

The feedback is vital to keep improving GP Power Tools based on what the Microsoft Dynamics GP community want and need.




The dialog only automatically opens for users with POWERUSER application security or the SQL Server sysadmin fixed server role. It will open two days after a new installation of GP Power Tools or immediately with an upgrade of GP Power Tools.

Once the dialog is opened, a selection of an action from the drop-down list is required to close the window and continue. You can decide to complete the Survey which will open the default web browser to the page, or you can decide to postpone the survey to a later time (next login, tomorrow, 30 days, or after installing the next build).

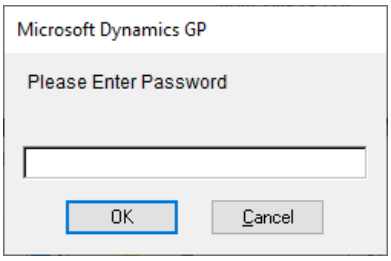
Once completed, the dialog will display the date and user details.

The dialog can also be opened manually by all users by selecting GP Power Tools Feedback Survey from the Routines section of the GP Power Tools Area Page or by selecting GP Power Tools Feedback Survey from the Options button drop list on the main GP Power Tools Logging Control window.

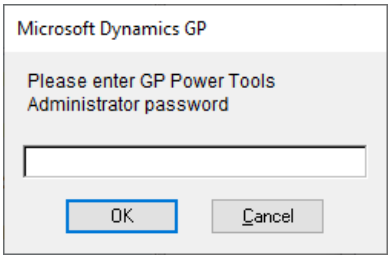
## Advanced Mode Access

To be able to access the Advanced Mode features of GP Power Tools, the current Microsoft Dynamics GP User ID will need to have either the SQL Server sysadmin fixed server role or membership of the db_owner role on the system database (DYNAMICS) and the current company database.

If the Microsoft Dynamics GP system password is configured, you will need to enter this password before the window will open.




Optionally, GP Power Tools can be configured to use its own Administrator password instead of the Microsoft Dynamics GP system password. If setup you will need to enter this Administrator password before the window will open.




Advanced Mode features are protected because they should be used only by Microsoft Dynamics GP system administrators, partner consultants or support engineers.

*Some Advanced Mode features allow direct access to data stored on the SQL Server. Other features can be used to disable functionality of Microsoft Dynamics GP.*

## GP Power Tools and the Web Client

GP Power Tools works with the Web Client, however some features are disabled as the functionality is not supported in the Web Client environment.

Below is a summary of features which are disabled or modified when running on the Web Client:

- *Accessing the tool is only via the Quick Links pane on the Home page.*
- *Macro Logging Mode is disabled.*
- *ScreenShot cannot capture bitmap images but can save or email System Status and other files.*
- *Changing Windows Titles to show User and/or Company is disabled.*
- *Preventing Windows opening outside the visible desktop is disabled.*
- *Changing background colors with Company Color Schemes is disabled.*
- *Microsoft Outlook Client email mode is not supported.*
- *Changing the launch file from Dictionary Control is disabled.*
- *Disabling VBA from Dictionary Control is disabled.*
- *Disabling Visual Studio Tools from Dictionary Control is disabled.*
- *Remembering position and size of windows is disabled.*
- *Using splitters on windows with two panes is disabled.*
- *Desktop Alerts show using a System Dialog.*
- *User Account Control (UAC) checks are disabled.*
- *Database Validation is not available when running on the web client.*
- *Use of Visual Studio dialogs such as MessageBox.Show() is not supported.*
- *Window Position Memory functionality is disabled.*
- *Launch File Configuration is not available when running on the web client.*
- *Virtual Fields are not available on the web client.*

# Chapter 3: System Module Features

This chapter includes the following sections:

- *Manual Logging Mode*
- *ScreenShot*
- *Send Email*
- *Calculator*
- *Dex.ini Settings*
- *Administrator Password Setup**
- *Logging Settings**
- *Email Settings**
- *Configuration Export/Import**
- *Configuration Maintenance**
- *Setup Backup and Restore**
- *Dictionary Assembly Generator Control**
- *Additional System Features*

** Advanced Mode Feature*

## Manual Logging Mode

The Microsoft Dynamics GP core application runs on the Dexterity runtime engine from which the following logging facilities are available:

### SQL Logging

SQL Logging tracks all communication between the Microsoft Dynamics GP client and the SQL Server. The default file in which the SQL communication is stored is named DEXSQL.LOG.

*The SQL Logging is tracked on a per workstation basis and will include information from more than one application session, if more than one session of Microsoft Dynamics GP is launched from the same application folder. This is normally the case for Terminal Server and Citrix installations.*

*Communication with the SQL Server using alternate methods of data access is not logged. For example; data access via Visual Basic for Applications (VBA) using ActiveX Data Objects (ADO) will not be captured by SQL Logging.*

### SQL Profile Tracing

SQL Profile Tracing can be used to log all activity at the SQL Server, including commands from inside Stored Procedures. The default file in which the SQL Profile Trace is stored is named Trace.trc.

*SQL Profile Tracing is not enabled until it has been setup using the SQL Profile Trace Settings window under the Logging Settings.*

*SQL Profile Tracing will capture all activity at the SQL Server for the current user in the DYNAMICS database and the current company database, so communication with the SQL Server using alternate methods of data access (as described above) will be logged.*

### Dexterity Script Logging

Dexterity Script Logging tracks all Dexterity event script, procedure and function calls, including the script hierarchy. The default file in which the script log is stored is named Script.log.

### Dexterity Script Profiling

Dexterity Script Profiling tracks the number of calls to each event script, procedure and function and how much time the calls have taken. It also tracks all table activity initiated by Dexterity and the time taken. The default file in which the script profile is stored is named Profile.txt.

### Macro Recording

Macro Recording captures all activity performed by the user at the User Interface. The recorded Macro can be replayed to repeat the actions, or opened in Notepad.exe for analysis. The default file in which the macro is stored is named Macro.mac.

*Macro Recording can be enabled using the Logging Settings window.*

*Macro Recording can only work when a single instance of Microsoft Dynamics GP is running on a workstation, or if multiple instances are running, Macro Recording will only work on the first instance launched.*

*Macro Recording is disabled when running on the Web Client.*

*Using any logging facility will create additional processing overhead for the application. Logging should only be used when actually looking to resolve an issue with the system.*

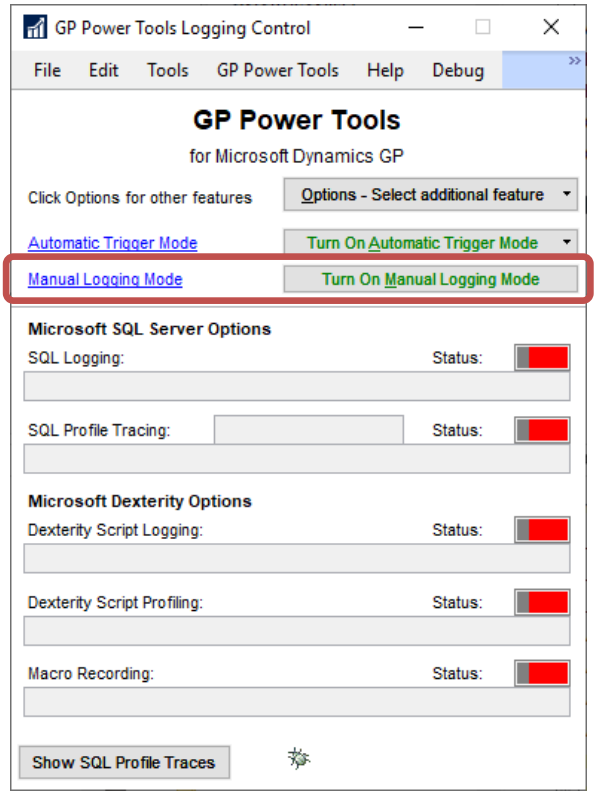
## Manual Logging Mode

By default, Manual Logging Mode will activate all of these logging options with a single mouse click without requiring the application to be restarted. The Dexterity runtime will continue to log activity in the application until stopped.

You can use the Logging Settings window to select which logging modes are enabled when using Manual Logging Mode.

To ensure that the log files are not overwritten, the User, Company and date and time information are appended to the default file name.

To start Manual Logging Mode, click on the Turn On button (highlighted below).




To stop Manual Logging Mode, click on the same button, now labeled as Turn Off.

Manual Logging can also be turned on using the Ctrl+Shift+F9 keyboard shortcut and turned off again using the Ctrl+Shift+F10 keyboard shortcut.

You may need to press and release the Alt key on the keyboard to allow the window menu bar to activate before the shortcut keys work.

*When using Manual Logging Mode to look at a specific issue (possible programming bug or performance problem), it is important to minimize the information captured in the logs to just the events directly related to the issue. To achieve this, request the user to perform all the actions in Microsoft Dynamics GP up to just prior to where the issue occurs. At this point, activate the manual logging and perform the action that exhibits the issue. Once the issue has occurred, stop the logging as soon as possible.*

The results of the logging can be found in the folder where GP Power Tools is storing its data files. The default location is the data subfolder beneath the Microsoft Dynamics GP application folder. The location can be changed from the default path using the Pathname location for Debugger Setup files, exports and logs option on the Dex.ini Settings windows (see section in this chapter).

The individual logs will be stored in the following files:

- GPPTools_<User>_<Company>.log or optionally GPPTools_<User>_<Company>_<Date>.log

  *This file will contain all the details of the actions performed by GP Power Tools including the names of the files created during the logging process. Any error or warning messages from GP Power Tools will also be logged to this file. Use the Logging Settings window add the optional date to the file name.*

- DEXSQL_<Date>_<Time>.LOG

  *These files will contain the SQL Logging results.*

- Trace_<User>_<Company>_<Date>_<Time>_<Mode>.trc

  *These files will contain the SQL Profile Tracing results.*

- Script_<User>_<Company>_<Date>_<Time>.log

  *These files will contain the Dexterity Script Logging results.*

- Profile_<User>_<Company>_<Date>_<Time>.txt

  *These files will contain the Dexterity Script Profiling results.*

- Macro_<User>_<Company>_<Date>_<Time>.mac

  *These files will contain the Macro Recording results.*

<User> will be substituted with the current User ID and <Company> will be substituted with the current Company ID code (InterCompany ID). <Date>_<Time> will contain the date and time at which the logging was started in the format YYYYMMDD_HHMMSS. <Mode> will be replaced with a letter A to E depending on the SQL Profile Trace mode used.

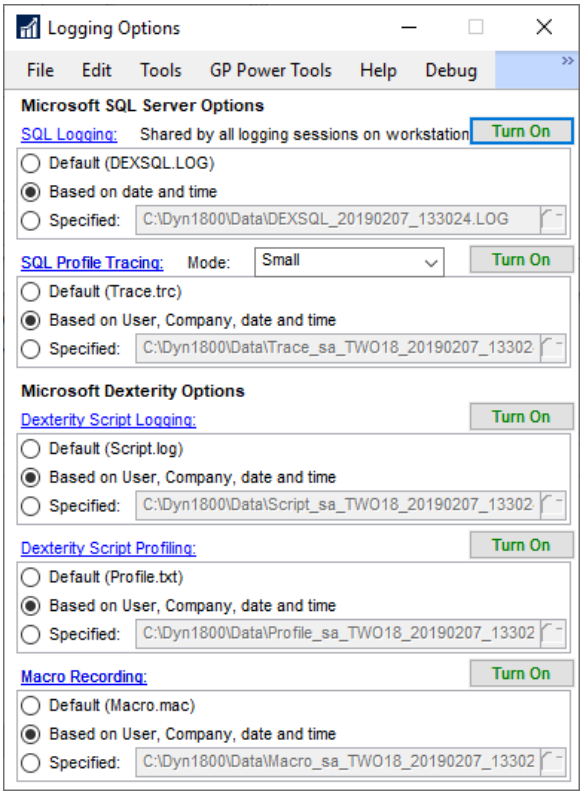
*When using the Dex.ini Setting to Start Logging on next startup, the file names used will not have a User ID or Company ID code as these will not be known until after login has completed.*

*There is an optional password which can be used to control access to Manual Logging Mode. It is set up using the Logging Password field on the General Tab of the Logging Settings window.*

## Individual Logging Control

Individual Logging Control allows each of the logging options to be controlled independently. To access the Individual Logging Control features it must be enabled via Logging Settings. Then you can click on the Logging Options button on the GP Power Tools main window.

The pathnames of the resulting files can be left as default, created automatically based on User, Company and date and time information or they can be manually specified.



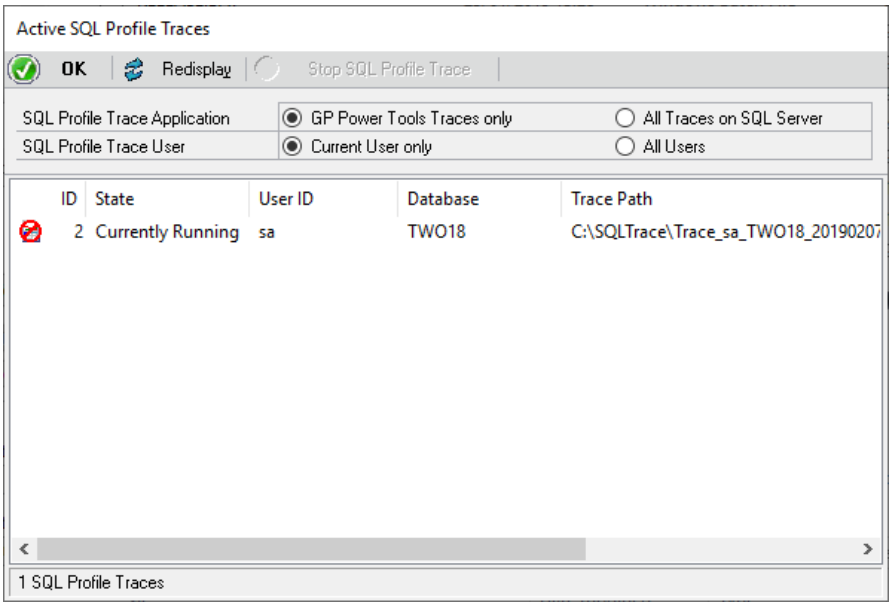

Turning all logging methods on using the Based on date and time and Based on User, Company, date and time is the same as using Manual Logging Mode.

*Access to Individual Logging Modes can be enabled using the Enable Individual Logging Modes option on the General Tab of the Logging Settings window.*

## SQL Profile Traces

Active SQL Profile Traces can be viewed by pressing the Show SQL Profile Traces button on the GP Power Tools main window. This will open the Active SQL Profile Traces window.

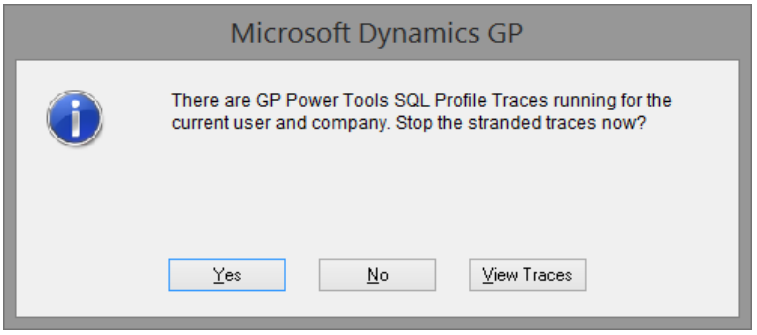
The window defaults to showing GP Power Tools Traces only for the SQL Profile Trace Application, and the Current User only for SQL Profile Trace User. A user with the sysadmin rights at the SQL Server level will be allowed to select All Traces on SQL Server or All Users modes.




Stranded SQL Profile Traces are traces created by GP Power Tools where the Microsoft Dynamics GP has unexpectedly terminated and left the trace running at the SQL Server. They can be stopped from this window by selecting the traces (use control and shift keys to multi-select) and then click Stop SQL Profile Trace.

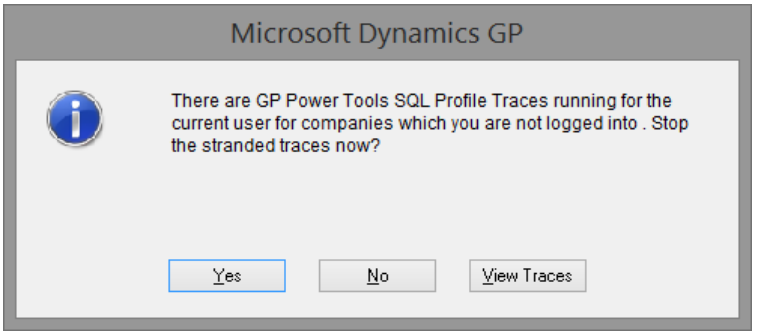
*The Show SQL Profile Traces button is enabled once SQL Profile Tracing has been enabled. For more information on setting up and enabling SQL Profile Tracing please see the section under the Logging Settings window or the section in the previous chapter.*

When logging into Microsoft Dynamics GP, if there are stranded traces for the current user and company, the following dialog will be displayed.




The user can select whether to stop the stranded traces, leave them running or open the Active SQL Profile Traces window.

Also when logging into Microsoft Dynamics GP, if there are stranded traces for the current user in other companies which the user is currently not logged into, the following dialog will be displayed.




Again the user can select whether to stop the stranded traces, leave them running or open the Active SQL Profile Traces window.

## ScreenShot

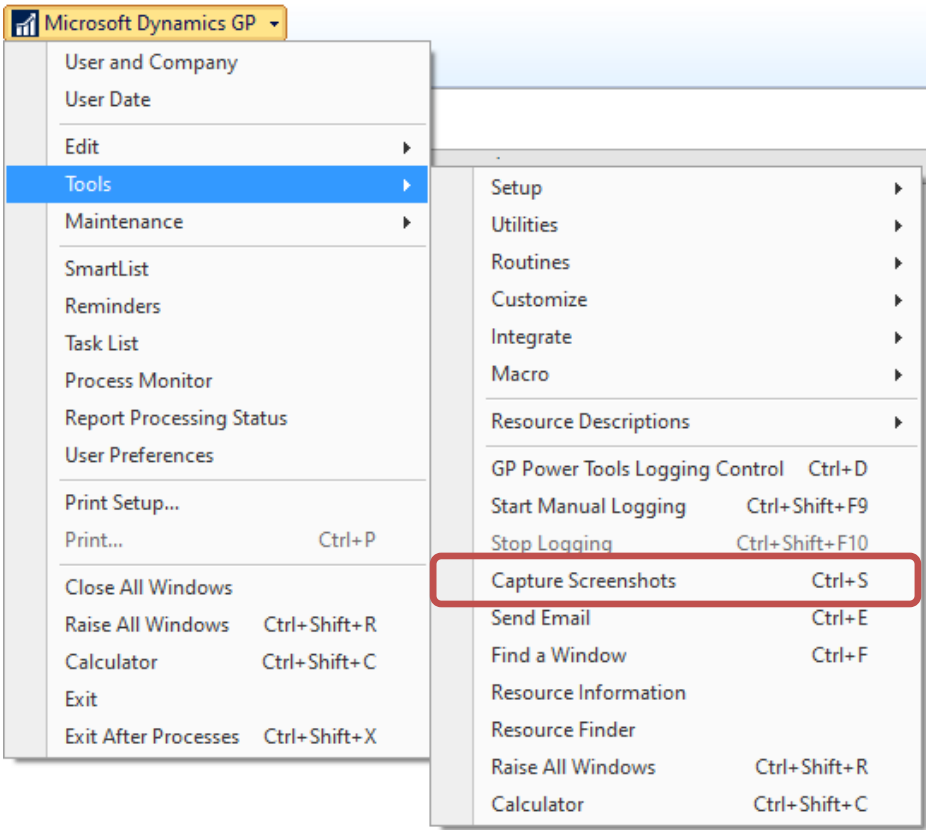
ScreenShot is a tool which can capture screenshots of all the open windows in the application as well as a System Status report and then either email or save the files.

Screenshot creates reduced color bitmaps (4 bits per pixel, 16 colors) to ensure that the size of the email is kept to a minimum. It can capture all open windows regardless of whether they are overlaid by other windows.

The System Status report contains information about the system including registration information, current login information, environmental information (such as operating system, database and ODBC versions), product information (including all version and build numbers) and a list of the attached screenshots.

You can open the GP Power Tools ScreenShot window by selecting Capture Screenshots from the Transactions section of the GP Power Tools Area Page or by selecting Capture Screenshots from the Options button drop-down list on the main window.

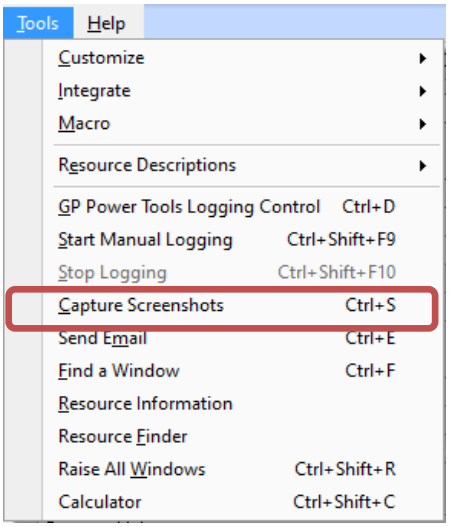
You can open it directly from the Tools menu underneath the Microsoft Dynamics GP menu (highlighted below). It also has the keyboard shortcut Ctrl+S assigned to it.



If the Standard Toolbar is displayed, you can launch ScreenShot from the Capture Screenshots button (highlighted below).

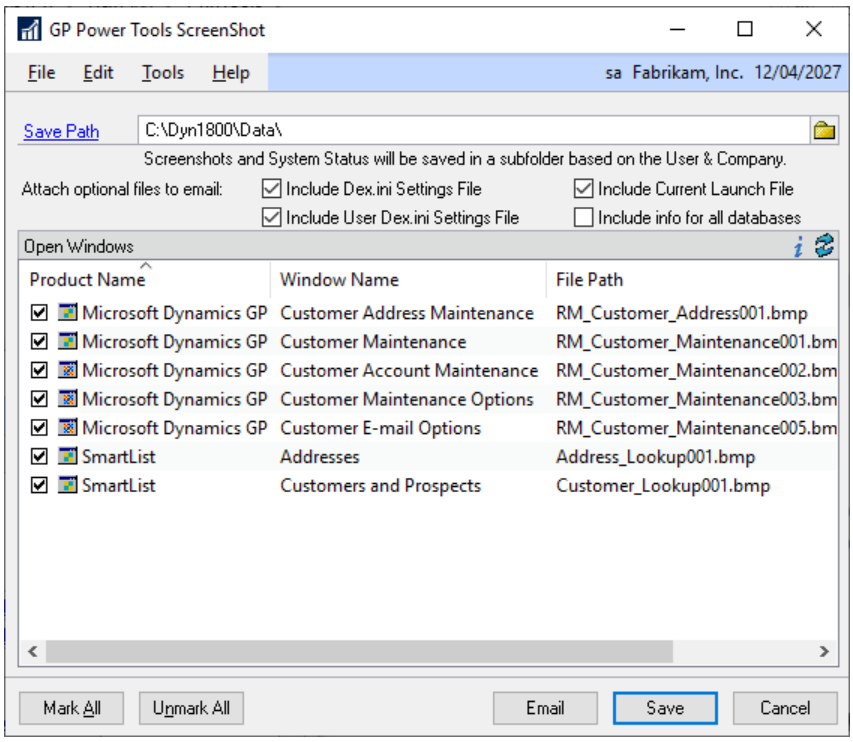
You can also use the Capture Screenshots option on Quick Links on the Home Page. When running on the Web Client, use the Quick Links on the Home Page to open Capture Screenshots as the other navigation options are not available.

In addition, Capture Screenshots is also found under the Tools menu on each individual window of Microsoft Dynamics GP (highlighted below).




You may need to press and release the Alt key on the keyboard to allow the window menu bar to activate before the shortcut keys work.

Once ScreenShot is activated, the following window will be displayed.




Below is a description of the individual fields on the window:

*Save Path*

This is the root path that will be used when saving screenshots. The actual path used will be a subfolder based on the user ID and company ID code.

*Include Dex.ini Settings File*

This checkbox tells ScreenShot whether to include the Global level Dex.ini settings file as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include User Dex.ini Settings File*

This checkbox tells ScreenShot whether to include the User level Dex.ini settings file as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include Current Launch File*

This checkbox tells ScreenShot whether to include the launch file, usually Dynamics.set, as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include info for all databases*

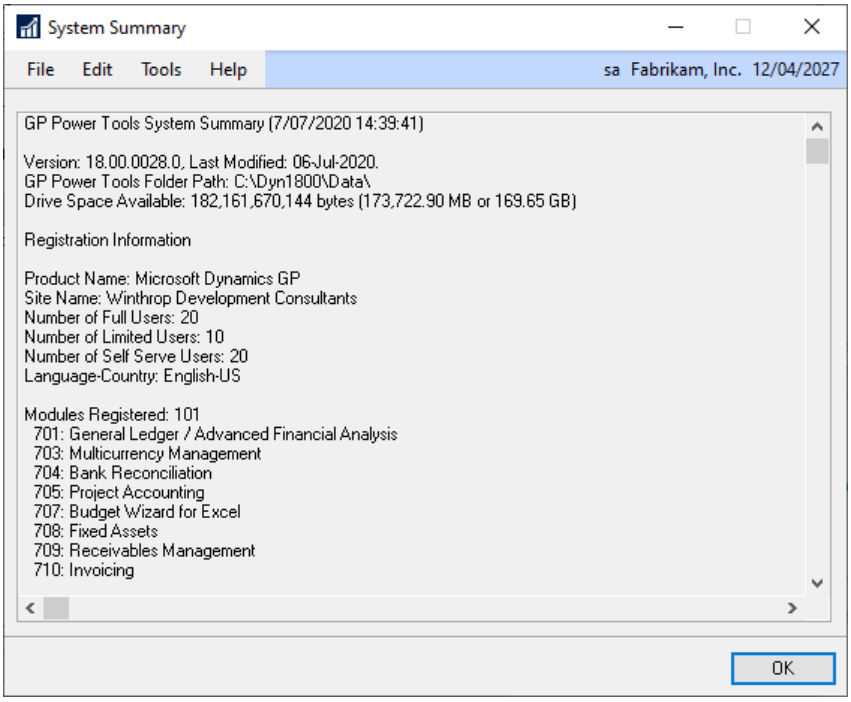
This checkbox tells ScreenShot whether to include information for all databases or just the system database and current company database in the System Status report. Not including information for all databases gives better performance on systems with many companies. The default setting for this checkbox can be set up in the Administrator Settings window.

### *Open Windows*

This is a list of windows currently open on the system. It is automatically updated when a form is open or closed. If you open a secondary window on a form, you can refresh the list with the Refresh Button. You can use the checkboxes to select which screenshots should be included.

### *Info Button*

This button can be used to preview the System Status report. You can use Ctrl-A to select the contents of the report and then Ctrl-C to copy it to the clipboard.




### *Refresh Button*

This button will refresh the window list with the currently open windows.

### *Mark All Button*

This button will select all windows (or all highlighted windows) to be emailed or saved. This button will be disabled when running on the Web Client.

### *Unmark All Button*

This button will de-select all windows (or all highlighted windows) so that individual windows can be selected. This button will be disabled when running on the Web Client.

### *Email Button*

This button will create an email with the selected screenshots and System Status report attached. The System Status will also be included as the body of the email. All that the user needs to do is add a recipient and click Send. The default email settings can be set up in the Email Settings window.

*Save Button*

This button will save the selected screenshots and System Status report to a folder based on the Save Path and the current user ID and company ID code.

*Cancel Button*

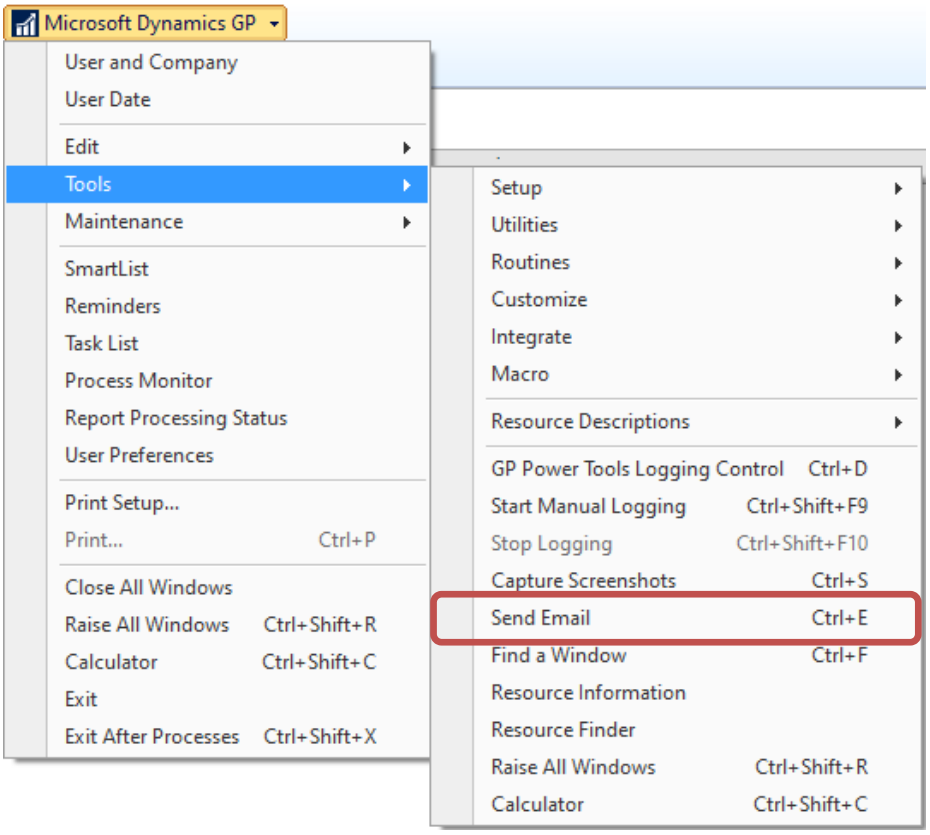
This button will close ScreenShot.

*When running on the Web Client, ScreenShot is unable to create the bitmap images and so this functionality is disabled..*

## Send Email

You can open the Send Email window by selecting Send Email from the Transactions section of the GP Power Tools Area Page or by selecting Send Email from the Options button drop list on the main window.

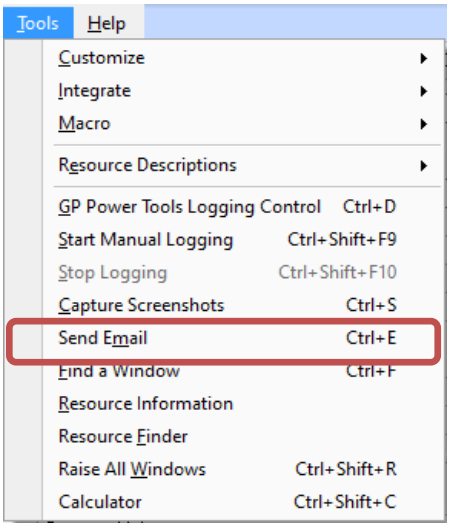
You can open it directly from the Tools menu underneath the Microsoft Dynamics GP menu (highlighted below). It also has the keyboard shortcut Ctrl+E assigned to it.




If the Standard Toolbar is displayed, you can launch Send Email from the Send Email button (highlighted below).

You can also use the Send Email option on Quick Links on the Home Page. When running on the Web Client, use the Quick Links on the Home Page to open Send Email as the other navigation options are not available.

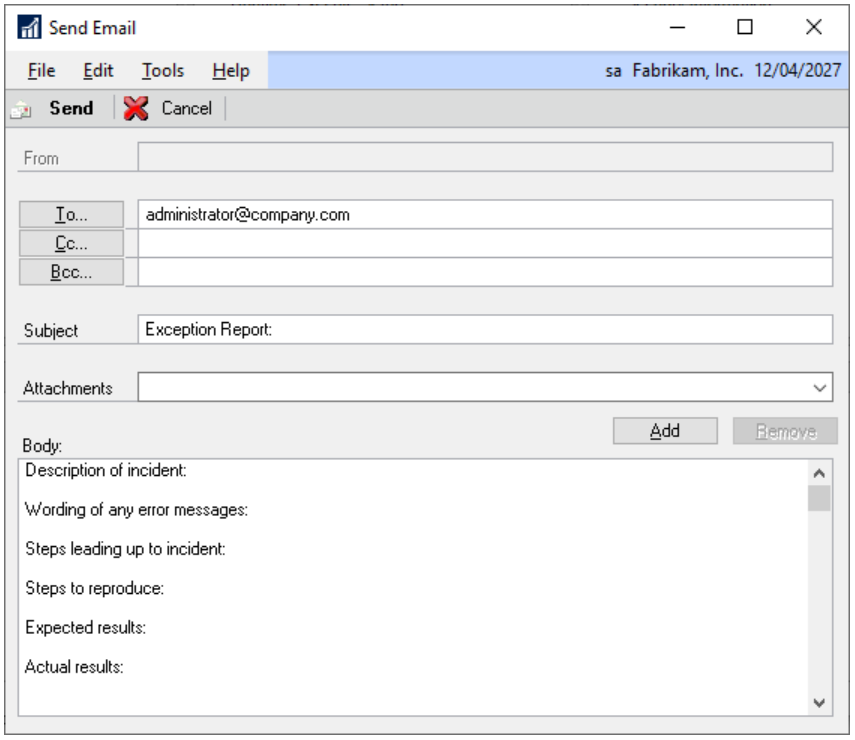
In addition, Send Email is also found under the Tools menu on each individual window of Microsoft Dynamics GP (highlighted below).




You may need to press and release the Alt key on the keyboard to allow the window menu bar to activate before the shortcut keys work.

The Send Email window can be used to create and send email messages from inside the Microsoft Dynamics GP application. This window will also appear to the user when other features in GP Power Tools are configured to send emails and the option to Preview emails is enabled in the Email Settings window.

The default email settings can be set up in the Email Settings window. This includes the Email address to use in the To address and the Default Subject and Default Body Text.



Email addresses can be in the following formats and multiple addresses should be separated by a semi-colon (;):

- name@domain.com
- Full Name<name@domain.com>
- Full Name (when in Microsoft Outlook mode only)

Below is a description of the individual fields on the window:

*From Field*

This is a single email address used as the sender's email when sending via SMTP mode. The default value is set up in the Email Settings window as the Sender's Email.

*To Field*

This is the list of email addresses to be used as the To value when sending the email. The To Button is available when a MAPI compliant email client is installed and allows the selection of addresses from an address book. The default value is set up in the Email Settings window as the Administrator Email.

*Cc Field*

This is the list of email addresses to be used as the Cc (Carbon Copy) value when sending the email. The Cc Button is available when a MAPI compliant email client is installed and allows the selection of addresses from an address book.

*Bcc Field*

This is the list of email addresses to be used as the Bcc (Blind Carbon Copy) value when sending the email. The Bcc Button is available when a MAPI compliant email client is installed and allows the selection of addresses from an address book.

*Subject*

This is the Subject line to be used when sending the email. The default value is set up in the Email Settings window as the Default Subject.

*Attachments*

This is a drop-down list containing the paths to the files to be attached when sending the email.

*Add Button*

This button opens a dialog to select a file to be added to the list of attachments.

*Remove Button*

This button removes the currently selected attachment from the list.

*Body*

This is the Body text to be used when sending the email. The default template can be set up in the Email Settings window as the Default Body Text.

*Send Button*

This button will process the email and send it. The transport protocols and other email settings can be set up in the Email Settings window.

### *Cancel Button*

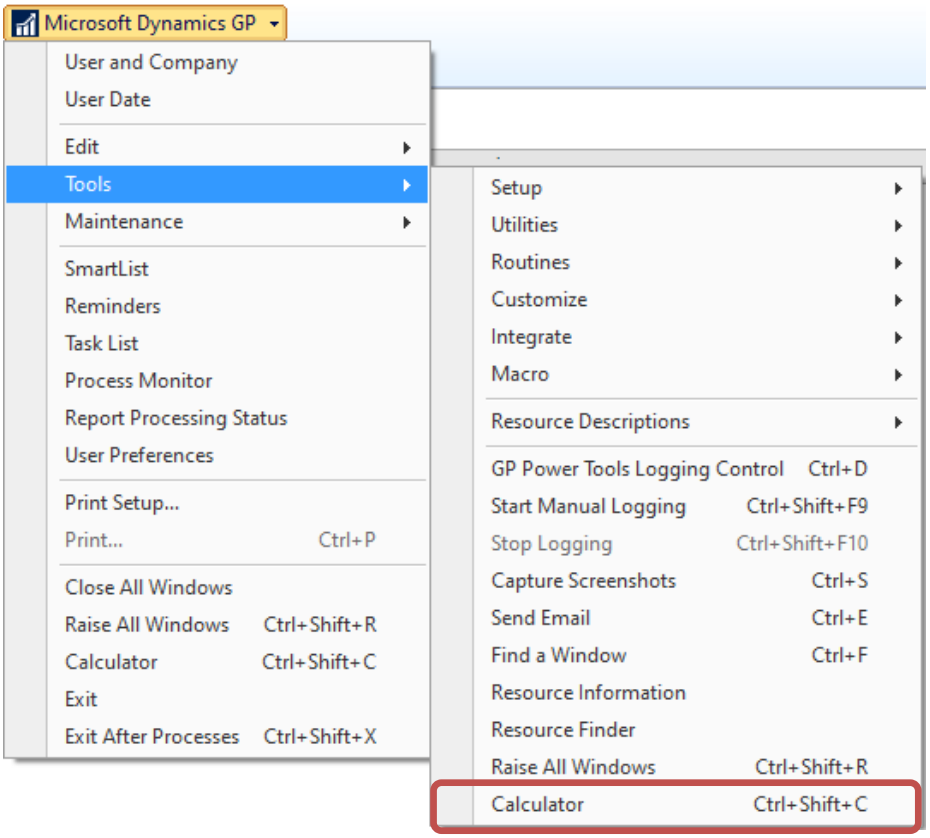
This button will abort the email and close the window.

*When the Send Email window is manually opened, it behaves as though Preview and Auto Send options are enabled in the Email Settings window. This is to ensure that the Send Email window is the only user interface seen when manually sending emails.*

## Calculator

You can open the Calculator window by selecting Calculator from the Transactions section of the GP Power Tools Area Page or by selecting Calculator from the Options button drop list on the main window.

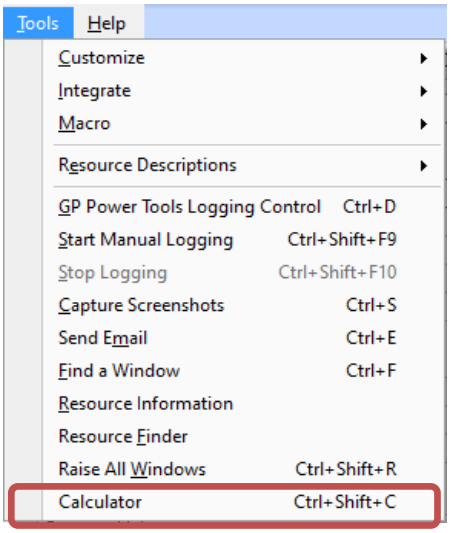
You can open it directly from the Tools menu underneath the Microsoft Dynamics GP menu (highlighted below). It also has the keyboard shortcut Ctrl+Shift+C assigned to it.




If the Standard Toolbar is displayed, you can launch Calculator from the Calculator button (highlighted below).

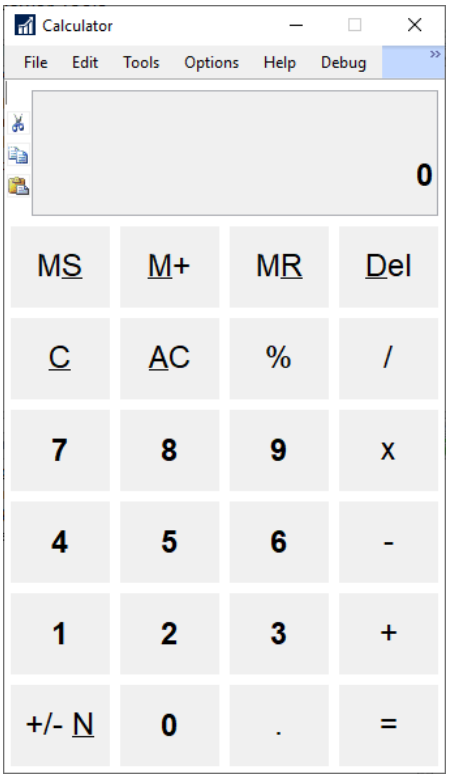
You can also use the Calculator option on Quick Links on the Home Page. When running on the Web Client, use the Quick Links on the Home Page to open the Calculator as the other navigation options are not available.

In addition, Calculator is also found under the Tools menu on each individual window of Microsoft Dynamics GP (highlighted below).




You may need to press and release the Alt key on the keyboard to allow the window menu bar to activate before the shortcut keys work.

The Calculator is a touch friendly standard calculator built directly into Microsoft Dynamics GP. It supports copying and pasting of values using the clipboard as well as memory functions. It is especially useful on the Web Client where access to a calculator app might not be possible.




*Use the highlighted letters for the Memory Store (S), Memory Add (M), Memory Recall (R), Delete (D), Clear (C), All Clear (A) and Negate (N) functions. It was not possible to get the Delete or Backspace keys on the keyboard working.*

## Dex.ini Settings

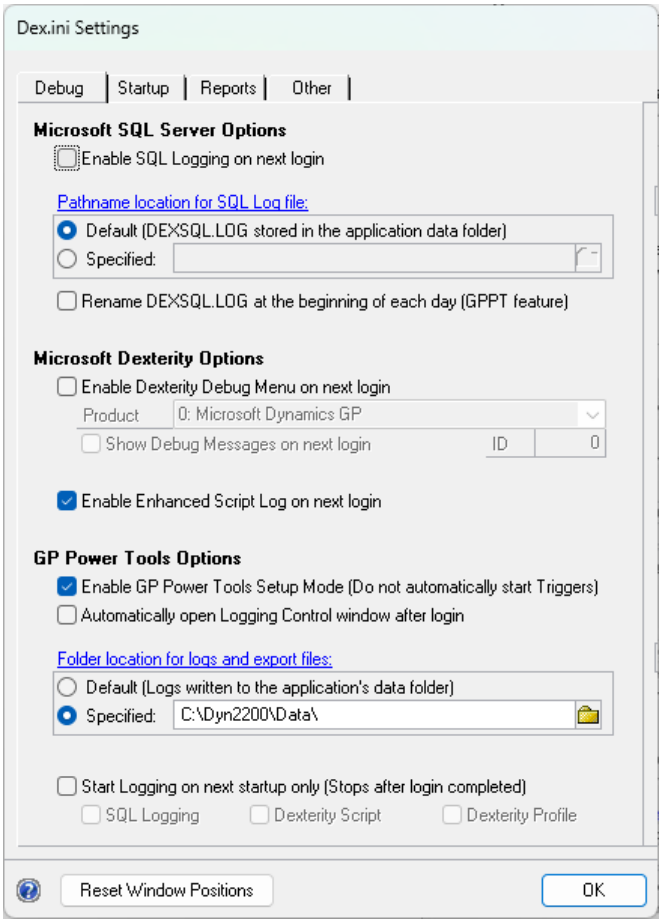
You can open the Dex.ini Settings window by selecting Dex.ini Settings from the Transactions section of the GP Power Tools Area Page or by selecting Dex.ini Settings from the Options button drop list on the main window.

The Dex.ini Settings window allows control of some system and GP Power Tools options which are stored in the Dex.ini file. It is divided into four tabbed sections.

*For Microsoft Dynamics GP 2013 onwards, all settings in this window are stored in the Global level Dex.ini with the exception of the Enable Debugger Setup Mode and Automatically open GP Power Tools main window after login options which are stored in the User level Dex.ini.*

## Debug Tab

The Debug tab contains settings related to the use of the logging and debugging features of Microsoft Dynamics GP as well as settings for GP Power Tools itself.



The following settings are available:

*Enable SQL Logging on next login*

This option will update the SQLLogSQLStmt and SQLLogODBCMessages Dex.ini settings to enable logging to the DEXSQL.LOG file on next login.

*Pathname location for SQL Log file*

This option will update the SQLLogPath Dex.ini setting to control the location of the DEXSQL.LOG file. This option controls where the log file is initially created. If using Automatic Trigger Mode or Manual Logging Mode, the log file will be renamed and possibly moved to a different folder.

*Rename DEXSQL.LOG at the beginning of each day*

This option is added by GP Power Tools to stop the DEXSQL.LOG file growing too large. It renames the log each day. It stores the date when it last renamed the file in the SQLLogRename Dex.ini setting in the format YYYYMMDD. This option should not be used with Automatic Trigger Mode.

*Enable Dexterity Debug Menu on next login*

This option will update the ScriptDebugger Dex.ini setting to control whether the Debug Menu will be available on next login.

*Dexterity Debug Menu Product*

This option updates the ScriptDebuggerProduct Dex.ini setting to control the default dictionary Product ID context for the Debug Menu.

*Show Debug Messages on next login*

This option updates the ShowDebugMessages Dex.ini setting to control whether internal debug message dialogs are displayed when the Debug Menu is enabled.

*If the Debug Menu is enabled, it is recommended that the Show Debug Messages option is not enabled for a production system. Having it enabled can cause additional dialogs to be displayed that could be confusing to users.*

*Enable Enhanced Script Log on next login*

This option updates the ScriptLogEnhanced Dex.ini setting to control whether the enhanced Dexterity Script Log features are enabled. Enabling this option adds time stamps and flagging of background processes to the script log. This option is enabled by default by GP Power Tools.

*Enable GP Power Tools Setup Mode*

Enabling this GP Power Tools option will prevent Triggers marked to Start Trigger Automatically on Login from starting. Setup Mode is designed to be used by consultants when setting up GP Power Tools for use at a customer's site. It uses the MBS_Debug_SetupMode Dex.ini setting.

*GP Power Tools Setup Mode should not be enabled for a production system. It is designed to only be used on test systems or support engineer or partner consultant's workstations.*

*Automatically open Logging Control window after login*
This option will make the GP Power Tools Logging Control main window open after a user logs in. It uses the MBS_Debug_AutoOpen Dex.ini setting.

*Folder location for logs and export files*
This option allows the location for any table exports and captured log files to be specified. It uses the MBS_Debug_Path Dex.ini setting.

*Start Logging on next startup only*
Enabling this GP Power Tools option will automatically start Manual Logging Mode on application startup. This enables the capture of the logs during the login process. This option will turn itself off after it has been used once. It uses the MBS_Debug_LogOnStartup Dex.ini setting.

*SQL Logging*
When using Logging on next startup, you can specify which logging options to enable. This option enables SQL Logging. It uses the MBS_Debug_LogOnStartup Dex.ini setting.

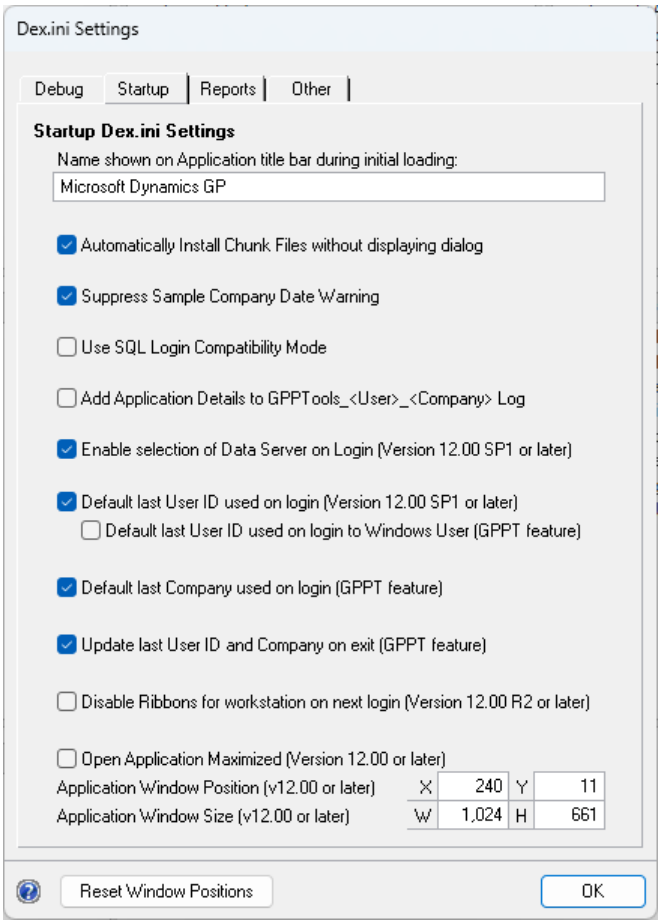
*Dexterity Script*
When using Logging on next startup, you can specify which logging options to enable. This option enables Dexterity Script Logging. It uses the MBS_Debug_LogOnStartup Dex.ini setting.

*Dexterity Profile*
When using Logging on next startup, you can specify which logging options to enable. This option enables Dexterity Profile Logging. It uses the MBS_Debug_LogOnStartup Dex.ini setting.

You can use the Reset Window Positions Button to clear the Dex.ini settings used for remembering the last window size, position and state for the GP Power Tools windows. Be sure all other GP Power Tools windows are closed when using this option.

## Startup Tab

The Startup tab contains settings related to the startup of Microsoft Dynamics GP.




The following settings are available:

*Name shown on Application title bar during initial loading*

This option will update the ApplicationName Dex.ini setting to control the name shown by the Dexterity Runtime title bar during application startup. Entering a value into this field will override the default application name of "Dexterity Runtime" while the application is launching. Once the application has launched, the title is updated with the product name as shown in the Dynamics.set launch file.

*Automatically Install Chunk Files without displaying dialog*

This option will update the AutoInstallChunks Dex.ini settings to allow chunk files to install without the user being prompted.

*Suppress Sample Company Date Warning*

This option will update the SAMPLEDATEMSG Dex.ini setting to allow Microsoft Dynamics GP to login to the Fabrikam sample company without displaying the date warning dialog.

*Use SQL Login Compatibility Mode*

This option will update the SQLLoginCompatibilityMode Dex.ini setting to allow Microsoft Dynamics GP to continue attempting to login using backwards compatible password encryption methods.

*If you continue to use SQL Login Compatibility Mode, a failed login attempt will register as four attempts at the SQL Server. This can prematurely lock out a user when an incorrect password is entered (when enforce password policy is enabled for the SQL Login and the SQL Native Client is used for the ODBC DSN configuration).*

*Add Application Details to GPPTools_<User>_<Company> Log*

This option will add an entry into the GPPTools_<User>_<Company>.log file each time a user logs into a company. It uses the MBS_Debug_LogAppDetails Dex.ini setting.

*Enable selection of Data Server on Login*

This option controls whether the Server drop-down list on the Login window is enabled. It uses the EnableServerDropDown Dex.ini setting.

*Default last User ID used on login*

This option controls whether the last User ID used is defaulted in on the Login window. It uses the DefaultLastUser Dex.ini setting.

*Default last User ID used on login to Windows User*

This option controls whether the last User ID used is defaulted in on the Login window to the current Windows User ID. It uses the DefaultLastUser Dex.ini setting.

*Default last Company used on login*

This option controls whether the last Company used is defaulted in on the Company Selection window. It uses the DefaultLastCompany Dex.ini setting.

*Update last User ID and Company on exit*

This option controls whether the last User ID and Company used are stored in the Dex.ini when exiting. This is useful when running multiple instances of Microsoft Dynamics GP, the last closed instance will record its settings rather than the last logged in settings. It uses the MBS_Debug_UpdateLastUserOnExit Dex.ini setting.

*Disable Ribbons for workstation on next login*

This option can disable Ribbons on the desktop client for the current workstation. It uses the EnableWCRibbons Dex.ini setting.

*Open Application Maximized on next login*

This option controls whether the application opens full screen for the current workstation. It uses the WindowMax Dex.ini setting.

*Application Window Position*

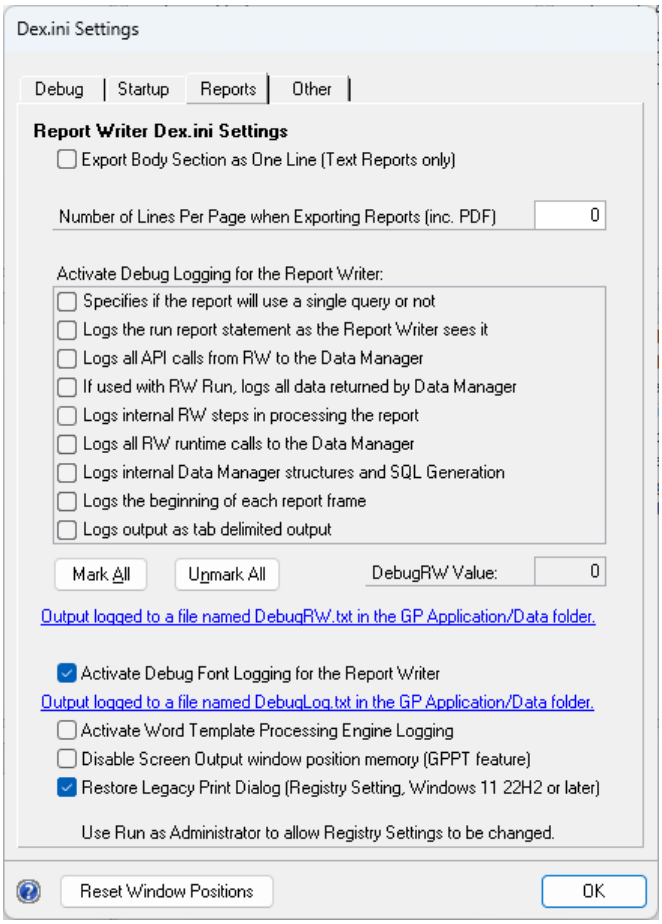
These options control the default application window position when not maximized for the current workstation. It uses the WindowPosX and WindowPosY Dex.ini settings.

*Application Window Size*

These options control the default application window size when not maximized for the current workstation. It uses the WindowWidth and WindowHeight Dex.ini settings.

## Reports Tab

The Reports tab contains settings related to the behavior and debugging of the Microsoft Dynamics GP Report Writer.




The following settings are available:

*Export Body Section as One Line*

This option will update the ExportOneLineBody Dex.ini setting to control how the body section on a text report is printed. This option can be used when creating reports to be exported as tab-delimited or comma-delimited text files.

*Number of Lines Per Page when Exporting Reports (inc. PDF)*

This option will update the ExportLinesPerPage and ExportPDFLinesPerPage Dex.ini settings to control the number of lines on a report page when the report is exported rather than printed to a file (including PDF files).

*Suggested values for this setting are 72 for A4 paper in portrait, 51 for A4 paper in landscape, 68 for US Letter paper in portrait and 52 for US Letter paper in landscape. Some trial and error testing might be required to find the best value.*

*Activate Debug Logging for the Report Writer*

These options will update the DebugRW Dex.ini setting to ask the Report Writer to output a debugging log to the file DebugRW.txt. The actual value written to the Dex.ini is shown in the DebugRW Value field.

*Mark All*

Use this button to activate all the Report Writer debug logging.

*Unmark All*

Use this button to turn off Report Writer debug logging.

*Activate Debug Font Logging for the Report Writer*

This option will update the DebugFonts Dex.ini setting enable logging of font selections made by the Report Writer. The results will be written to a DebugLog.txt file. For more information see Knowledge Base (KB) article 870341:

http://support.microsoft.com/kb/870341

*Activate Word Template Processing Engine Logging*

This option will update the TPELogging and the KeepTemplateTempFiles Dex.ini settings to log the workings of the Template Processing Engine (TPE). The following files will be created in the %TEMP% folder: the TemplateProcessing*.txt file, the document file and the template file.

*Disable Screen Output window position memory*

This option can be used to disable the window position memory for the Report Writer Screen Output window. It will update the MBS_Debug_DisableScreenOutputMemory and the MBS_Debug_WinScreenOutput Dex.ini settings.

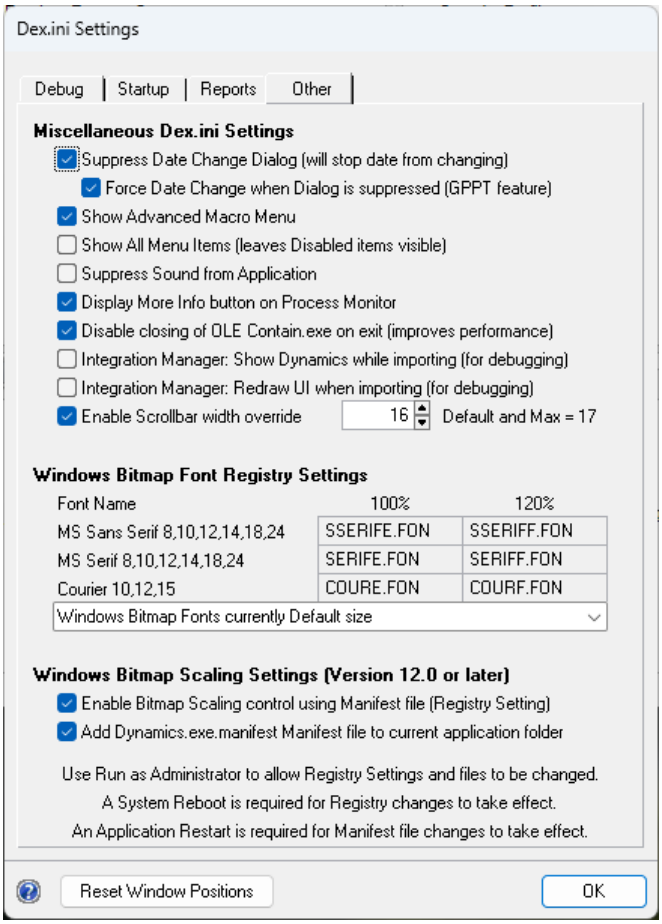
*Restore Legacy Print Dialog (Registry Setting, Windows 11 22H2 or later)*

This option restores the legacy print dialog which users are expecting rather than using the modern dialog added in Windows 11 22H2 release.

*From Build 30 onwards the Legacy Print Dialog is restored during the installation of GP Power Tools. This is because it has features that Microsoft Dynamics GP uses which are not available on the modern dialog.*

## Other Tab

The Other tab contains other miscellaneous settings for use with Microsoft Dynamics GP.




The following settings are available:

*Suppress Date Change Dialog*

This option will update the SuppressChangeDateDialog Dex.ini setting to prevent the dialog to change the User Date from being displayed at midnight. Using this option will also stop the date from being changed in Microsoft Dynamics GP (see option below).

*Force Date Change when Dialog is suppressed*

This option will update the SuppressChangeDateForce Dex.ini setting to force the User Date to be changed at midnight when the dialog is suppressed with the above option. This functionality can also be overridden using the option that is part of Automatic Logout in the Administrator Settings window.

*Show Advanced Macro Menu*

This option will update the ShowAdvancedMacroMenu Dex.ini setting to enable the Advanced Macro Menu from the Tools >> Macro menu.

*Show All Menu Items*

This option will update the ShowAllMenuItems Dex.ini setting to show all menu items, even when the module is not installed, not registered or access has been denied.

*Suppress Sound from Application*

This option will update the SuppressSound Dex.ini setting to suppress all sound from Microsoft Dynamics GP.

*Display More Info button on Process Monitor*

This option will update the QueueMoreInfo Dex.ini setting to display the More Info button on the Process Monitor window (Microsoft Dynamics GP >> Process Monitor).

*Disable closing of the OLE Contain.exe on exit*

This setting stops the application from attempting to close the OLE Contain.exe program on exit. It can improve performance when exiting the application. It updates the OLEClose Dex.ini setting.

*Integration Manager: Show Dynamics while importing (for debugging)*

This setting helps with debugging Integration Manager by displaying the application during the imports. It updates the ShowDynamics Dex.ini setting.

*Integration Manager: Redraw UI when importing (for debugging)*

This setting helps with debugging Integration Manager by forcing the UI to redraw during the imports. It updates the DUIRedraw Dex.ini setting.

*Enable Scrollbar width override*

This option will update the MaxSWScrollbarSize Dex.ini setting to override the width of scrollbars in Microsoft Dynamics GP. This can be helpful when changing DPI settings makes the scrollbars too wide, so that they cover up the contents of fields. The maximum value for this setting is the default value of 17.

*Windows Bitmap Font Registry Settings*

This option will attempt to change the registry to update the font files used for bitmap fonts under Windows 7 and later. These settings are initially created when the operating system is first installed and are not changed when changing the DPI setting for the system. If the fonts in the Microsoft Dynamics GP windows are not being displayed at the correct size, use this option to change the sizes.

*Windows Bitmap Scaling Settings*

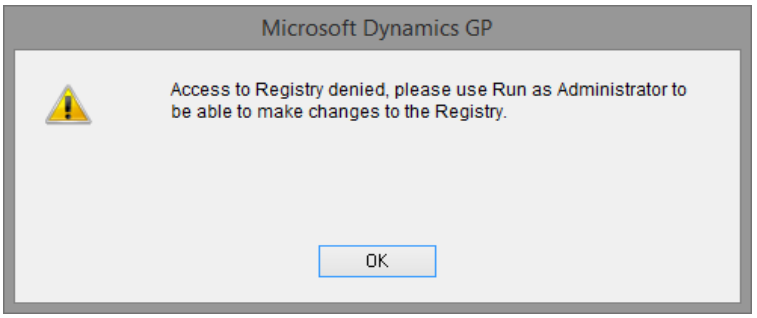
These options will attempt to update the registry to enable the Bitmap Scaling functionality of Windows 8 or later and create a Manifest file to enable Bitmap Scaling for the current instance of the Microsoft Dynamics GP application.

Bitmap Scaling can be used to tell the operating system that an application does not automatically handle high DPI settings (anything greater than 100%). The result is that the application is rendered off screen at 100% and then scaled to the correct percentage on the display. Using a DPI setting on the monitor which is a multiple of 100% (such as 200% or 300%) will produce the clearest images with no blurring of fonts, otherwise expect some fuzziness.

*From Build 29 onwards the Windows Bitmap Scaling Settings will be enabled during the installation of GP Power Tools while permissions are elevated. There is no harm enabling the settings on a system using 100% DPI as it will have no effect but will take effect if using a higher DPI setting.*

*A restart of the operating system is required for these settings to take effect.*

*On an operating system with User Account Control (UAC) enabled, Registry changes are only allowed if the application has been launched using Run as Administrator. If access to the registry is denied the following warning will be displayed:*



## Administrator Password Setup

You can open the Administrator Password Setup window by selecting Administration Password Setup from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Administrator Password Setup from the Options button drop list on the main window. You will also need to be added the GP POWER TOOLS PASSWORD Security Role to access this window. This is an Advanced Mode feature.

The Administrator Password Setup window can be used to specify a separate password to the System Password to be used to before an Advanced Mode feature window can be opened. This would allow users to have access to GP Power Tools administrator level windows without having to provide them with the System Password. Note that the user will still need the appropriate application level and SQL Server level security access.


The following is a description of the individual fields on the window:

*Use separate password instead of System Password*

This checkbox tells GP Power Tools to use a separate GP Power Tools Administrator Password instead of the System Password when opening Advanced Mode feature windows.

*Don't ask for users who have access to this window*

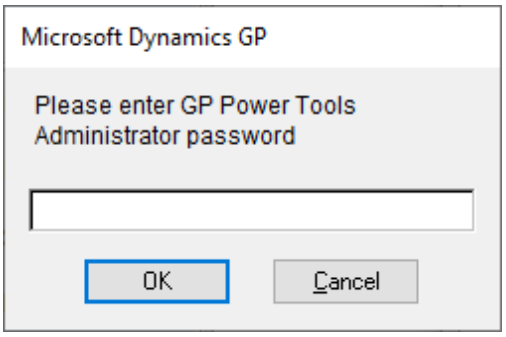
This checkbox tells GP Power Tools to not ask for the separate GP Power Tools Administrator Password for users who have application security access to the Administrator Password Setup window, except window opening this window. Add the GP POWER TOOLS PASSWORD Security Role to a user for access to this window.

*Password Fields*

These fields allow the separate GP Power Tools Administrator Password to be changed.

*Challenge 'sa' user with Administrator password on login*

This checkbox will force the 'sa' user to have to correctly enter the Administrator password before they can select a company during login. If they fail to enter the password, they will be unable to complete logging in. This feature is designed to prevent IT staff from easily accessing the Microsoft Dynamics GP application.

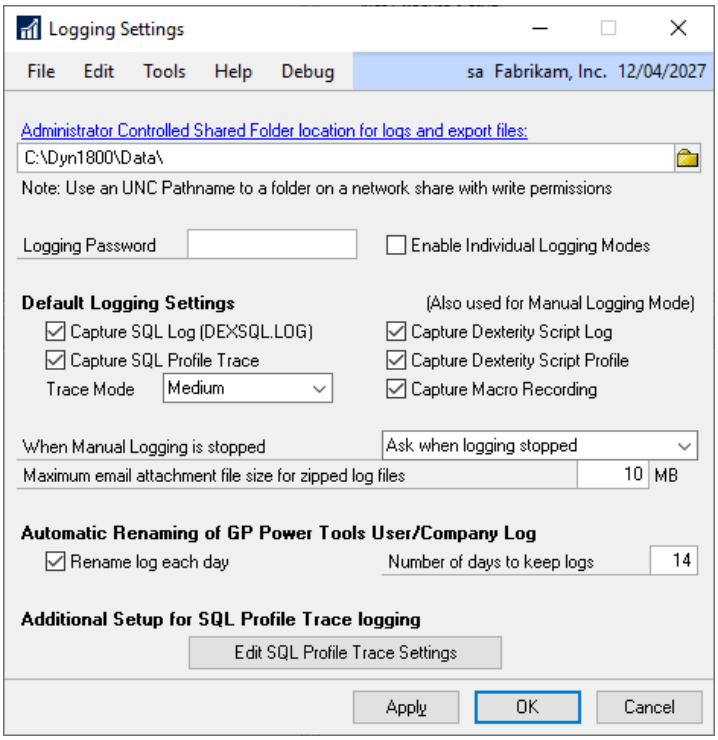



*If you enable the separate GP Power Tools Administrator Password, but don't actually set a new password, you can disable GP Power Tools asking for a password, without having to remove the System Password.*

## Logging Settings

You can open the Logging Settings window by selecting Logging Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Logging Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

The Logging Settings window can change the settings used with the GP Power Tools logging features.




The following is a description of the individual fields on the window:

*Administrator Controlled Shared Folder Location for logs and export files.*

You can select a folder in a shared location for all logs and export files to be written to. This setting is automatically rolled out to all workstations. Therefore avoiding the need to visit individual workstations to change the Pathname location for Debugger Setup files, exports and logs setting in the Dex.ini Settings window manually. It will update the MBS_Debug_Path Dex.ini setting on login.

*The Administrator Controlled Shared Folder Location Setting is stored in the syUserDefaults (SY01402) table in the DYNAMICS SQL Database. On login, the setting is checked and the Dex.ini setting on the current workstation is updated if necessary. The pathname can be specified using a UNC path in the format \\Server\Share\Folder\.*

*Logging Password*

You can specify an optional password to be requested before Manual Logging Mode can be enabled.

*Enable Individual Logging Modes*

You use this option to enable Individual Logging Control. By default this option is disabled which hides the Logging Options button on the GP Power Tools main window

*Capture SQL Log*

You can select which of the logging modes to enable, this option enables the SQL Logging when Manual Logging Mode is used.

*Capture SQL Profile Trace*

You can select which of the logging modes to enable, this option enables the SQL Profile Tracing when Manual Logging Mode is used.

*SQL Profile Tracing is not enabled until it has been setup using the SQL Profile Trace Settings window.*

*SQL Profile Trace Mode*

When using SQL Profile Tracing, you can use this option to select the type of SQL Profile Trace created. You can select between Small, Medium, Large and Performance. The Other mode can be used in conjunction with a customized MBS_SQL_Tracing_API_5 stored procedure in the DYNAMICS database.

*Capture SQL Log*

You can select which of the logging modes to enable, this option enables the SQL Logging when Manual Logging Mode is used.

*Capture Dexterity Script Log*

You can select which of the logging modes to enable, this option enables the Dexterity Script Logging when Manual Logging Mode is used.

*Capture Dexterity Script Profile*

You can select which of the logging modes to enable, this option enables the Dexterity Script Profiling when Manual Logging Mode is used.

*Capture Macro Recording*

You can select which of the logging modes to enable, this option enables the Macro Recording when Manual Logging Mode is used.

*Macro Recording can only work when a single instance of Microsoft Dynamics GP is running on a workstation, or if multiple instances are running, Macro Recording will only work on the first instance launched.*

*When Manual Logging is stopped*

You can select whether you want to email a zipped archive file of the logs captured by Manual Logging Mode when the logging is stopped. You can select to email logs automatically, or to ask before emailing logs.

*Maximum email attachment file size for zipped log files*

Use this option to select the maximum size allowed when emailing the zipped archive file containing the log files captured by Manual Logging Mode.

*The maximum email attachment file size would be limited by the maximum attachment size allowed by the email services being used. Please contact the administrator of the email system to check what the maximum size allowed is.*

*Rename log each day*

Select this setting to create a new GP Power Tools Log file for each user and company each day. This avoids the issue where the single undated file can get too large over time.

*Number of days to keep logs*

Use this option to control how many days' worth of GP Power Tools logs is kept before they are automatically removed. This prevents the logging folder from getting filled up with too many files.

*Edit SQL Profile Trace Settings*

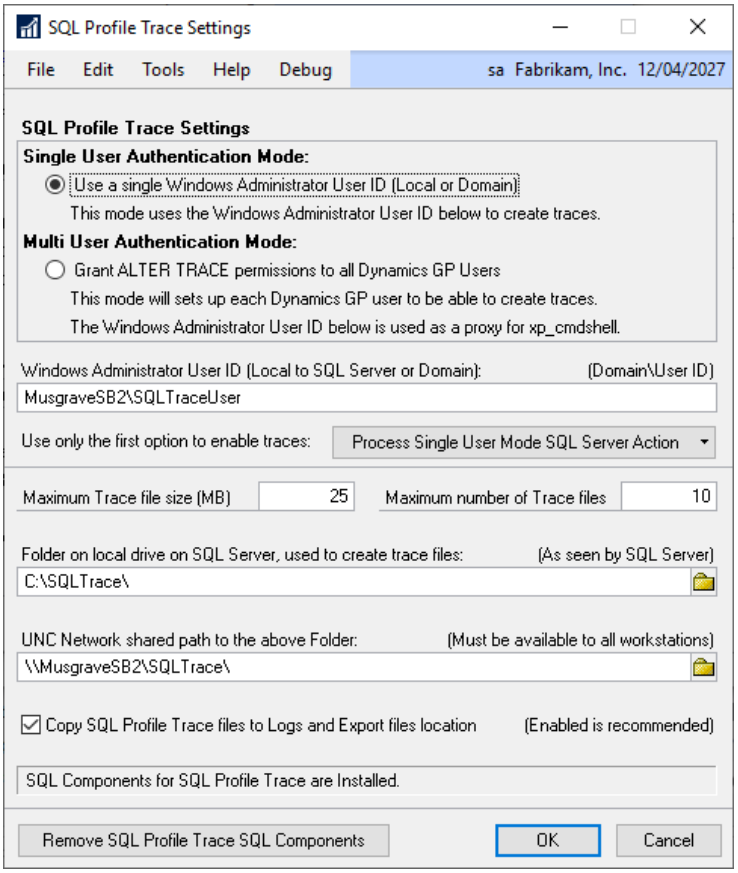
This button will open the SQL Profile Trace Settings window (see section below).

## SQL Profile Trace Settings

The SQL Profile Trace Settings window contains all the options to enable SQL Profile Tracing and create the database objects needed.

The SQL Profile Tracing functionality of GP Power Tools creates a series of SQL Stored Procedures in the DYNAMICS system database:

- MBS_SQL_Tracing_API
- MBS_SQL_Tracing_API_1 (Small)
- MBS_SQL_Tracing_API_2 (Medium)
- MBS_SQL_Tracing_API_3 (Large)
- MBS_SQL_Tracing_API_4 (Performance)
- MBS_SQL_Tracing_API_5 (Other)
- MBS_SQL_Tracing_Read
- MBS_SQL_Tracing_Version

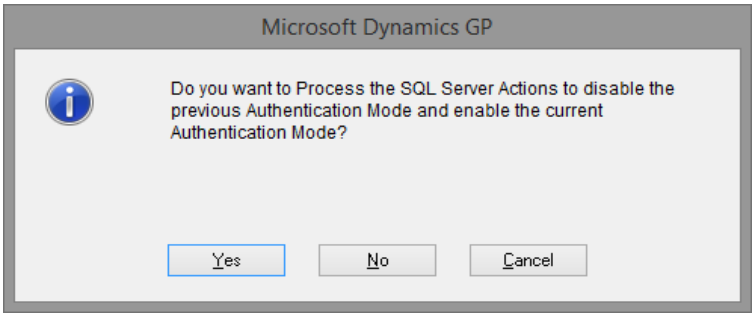


The following is a description of the individual fields on the window:

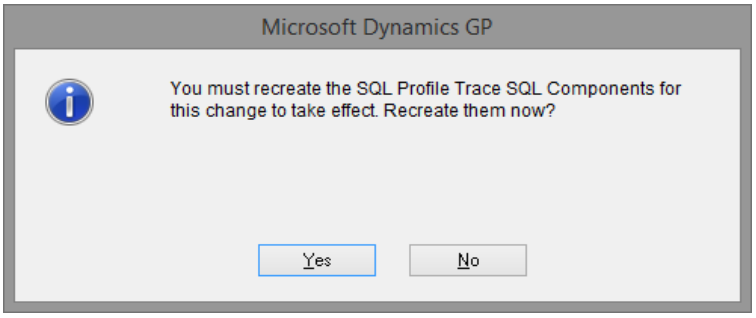
### *Single User Authentication Mode*

Single User Authentication Mode uses a single Windows user to create the SQL Profile Traces. This is the preferred Authentication Mode as it does not require individual users to have their privileges elevated.

If the Authentication Mode is already enabled and you change the setting, you will receive a dialog to process the necessary changes at the SQL Server. It is recommended that you allow the system to make the changes.



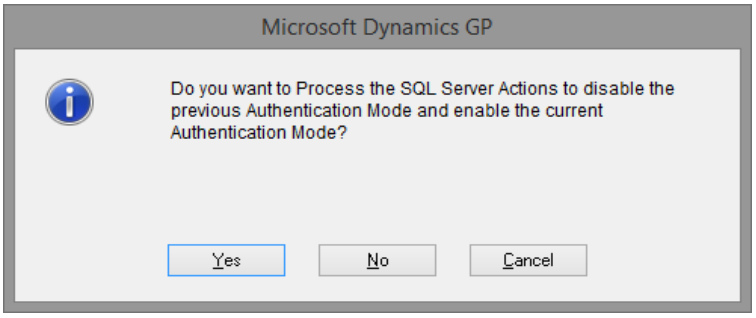

If the SQL Profile Trace SQL Components are already created and you change the setting, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.



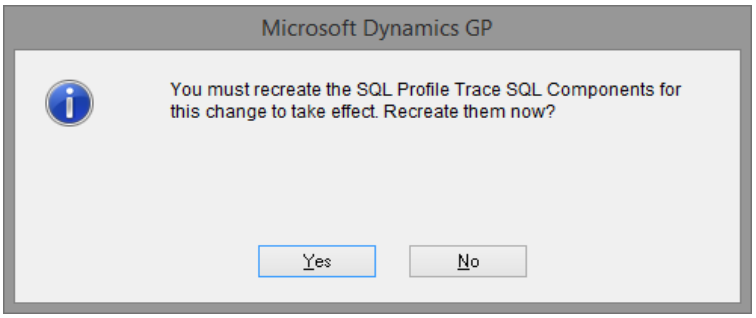

### *Multi User Authentication Mode*

Multi User Authentication Mode uses the individual Dynamics GP users to create the SQL Profile Traces and only uses the Windows user as a proxy for the xp_cmdshell command. Using this mode will elevate individual users' rights to allow them to create traces.

If the Authentication Mode is already enabled and you change the setting, you will receive a dialog to process the necessary changes at the SQL Server. It is recommended that you allow the system to make the changes.




If the SQL Profile Trace SQL Components are already created and you change the setting, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.



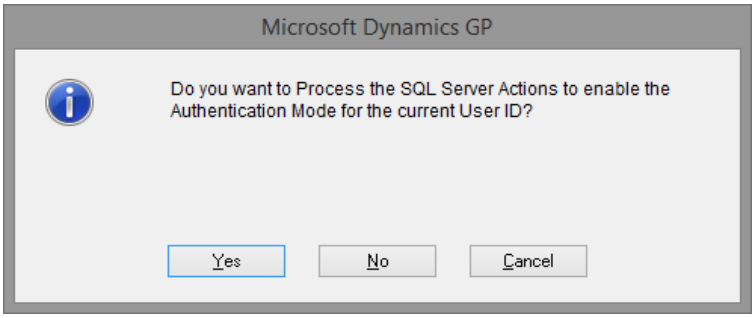

### *Windows Administrator User ID*

Depending on Authentication Mode, this Windows User ID is used to create traces and/or as a proxy for the xp_cmdshell command.

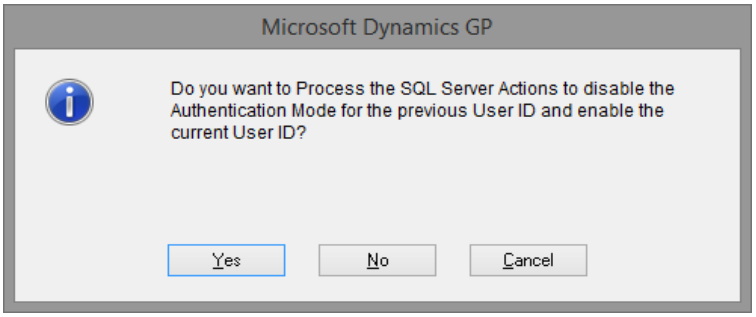
*The user can be either a local user on the SQL Server machine or a domain user. The user must be added to the local administrator group on the SQL Server machine. It is recommended that the password for the user is set to not expire.*

*The user does not need to be manually added to SQL Server,* GP Power Tools *will perform that step*.

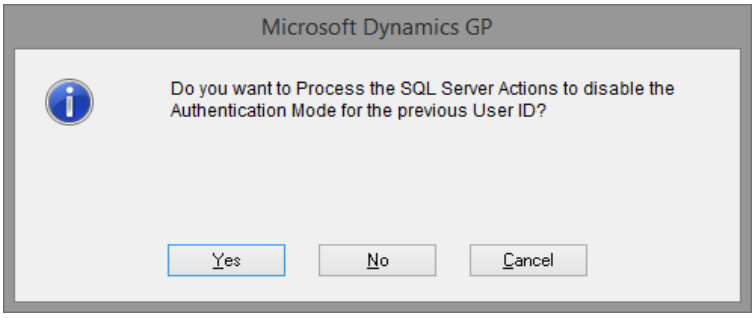
When you enter the User ID, you will receive a dialog asking to process the steps to enable the Authentication mode. It is recommended that you allow the system to make the changes.



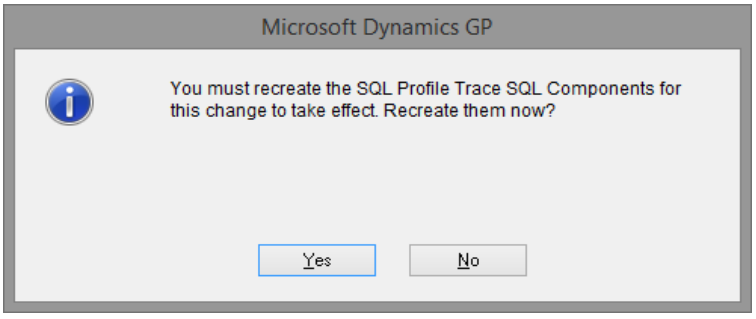
If the Authentication Mode is already enabled and you change the User ID, you will receive a dialog to process the necessary changes at the SQL Server. It is recommended that you allow the system to make the changes.




If you remove the User ID, you will receive a dialog asking to process the steps to disable the Authentication mode. It is recommended that you allow the system to make the changes.




If the SQL Profile Trace SQL Components are already created and you change the User ID, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.



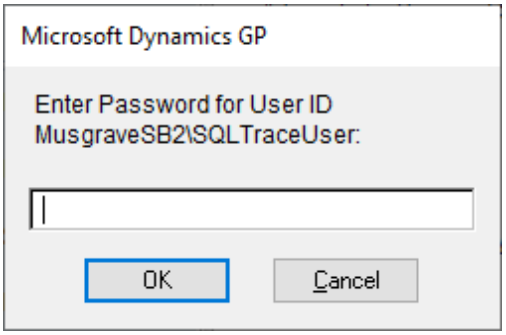
### *Process Single User Mode SQL Server Action*

There are seven setting changes required on SQL Server to allow Single User Authentication Mode to work. This button allows the steps to enable and disable the settings to be run individually or as one action.

As the system already prompts for these actions to be executed automatically, you would not normally need to manually run the actions using this button.

*If a new user is added to Microsoft Dynamics GP, you will need to run the Grant IMPERSONATE permission to all users option again to allow the new user to be able to create traces.*

The Enable xp_cmdshell proxy account with User ID option will ask for the password for the Windows Administrator User ID.




*The password is not validated at this time. If it is not entered correctly, it will prevent the SQL Profile Trace File being copied to the Debugger Settings folder when the trace is stopped. The error will show in the GPPTools_<User>_<Company>.log file.*

As each step is processed a Desktop Alert is displayed to show that the actions completed.

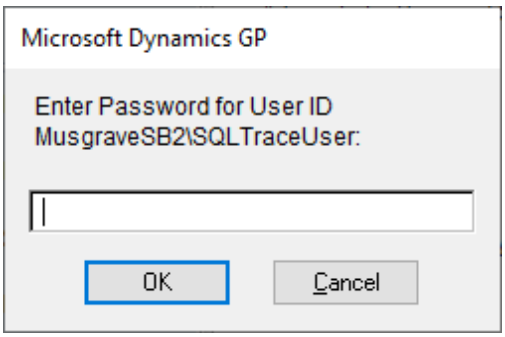
### *Process Multi User Mode SQL Server Action*

There are four setting changes required on SQL Server to allow Multi User Authentication Mode to work. This button allows the steps to enable and disable the settings to be run individually or as one action.

As the system already prompts for these actions to be executed automatically, you would not normally need to manually run the actions using this button.

*If a new user is added to Microsoft Dynamics GP, you will need to run the Grant IMPERSONATE permission to all users option again to allow the new user to be able to create traces.*

The Enable xp_cmdshell proxy account with User ID option will ask for the password for the Windows Administrator User ID.




*The password is not validated at this time. If it is not entered correctly, it will prevent the SQL Profile Trace File being copied to the Debugger Settings folder when the trace is stopped. The error will show in the GPPTools_<User>_<Company>.log file.*

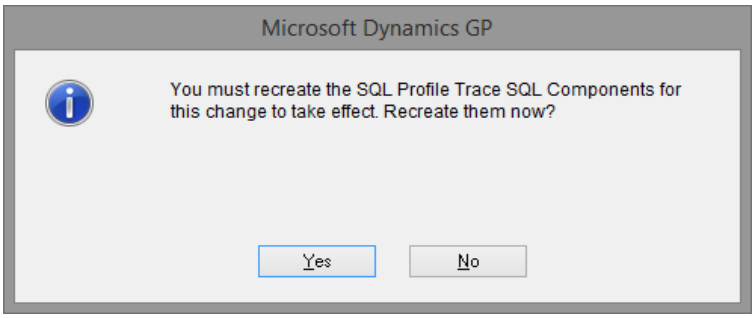
As each step is processed a Desktop Alert is displayed to show that the actions completed.

### *Maximum Trace file size*

Use this setting to control the maximum size a SQL Profile Trace file can get to before a new file is created. The default value for this field is 25 MB.

If you set the field back to zero, it will restore the default values for Maximum Trace file size and Maximum number of Trace files.

If the SQL Profile Trace SQL Components are already created and you change this setting, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.

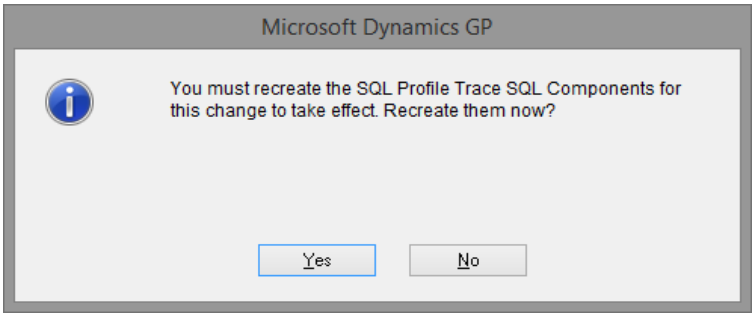



### *Maximum number of Trace files*

Use this setting to control the number of trace files created by the SQL Profile Trace. As the trace file reaches the Maximum Trace file size a new trace file will be created with a numbered suffix added to the filename. This setting controls how many of the individual trace files are kept and will delete the oldest trace files as new ones are created. The default value for this field is 10.

If you set the field back to zero, the trace will only create a single file which will grow until the trace is stopped.

If the SQL Profile Trace SQL Components are already created and you change this setting, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.

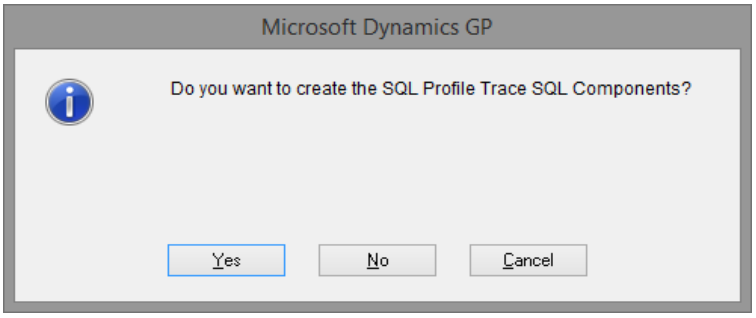



*Folder on local drive on SQL Server*

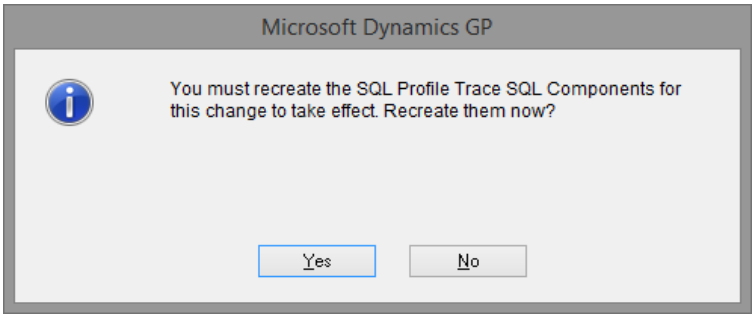
This is the path to a folder that is local to SQL Server that is to be used as a temporary location for SQL Profile Trace files while they are being created.

*The folder must use a path that is valid as seen from the SQL Server machine. All Microsoft Dynamics GP Users as well as the Windows Administrator User ID must have Full Control rights to this folder.*

When you enter the path, you will receive a dialog asking to create the SQL Profile Trace SQL Components (stored procedures). It is recommended that you allow the system to make the changes.




If the SQL Profile Trace SQL Components are already created and you change the path, you will receive a dialog to recreate them. It is recommended that you allow the system to make the changes.



*UNC Network shared path to above Folder*

This is the path to the local folder on the SQL Server from the previous field once it has been shared on the network.

*The folder must be shared so that all Microsoft Dynamics GP Users as well as the Windows Administrator User ID have Full Control rights to this folder.*

This path is used after the SQL Profile Trace is created to copy the trace files from the temporary location on the SQL Server to the Debugger Settings location.

*Copy SQL Profile Trace files to Logs and Export files location*

This checkbox can be used to control where the SQL Profile Trace files are copied from the temporary location on the SQL Server to the Logs and Export files location.

It is recommended that this setting is enabled.

*Create SQL Profile Trace SQL Components*

This button can be used to manually create the SQL Profile Trace SQL Components (stored procedures) on the SQL Server.

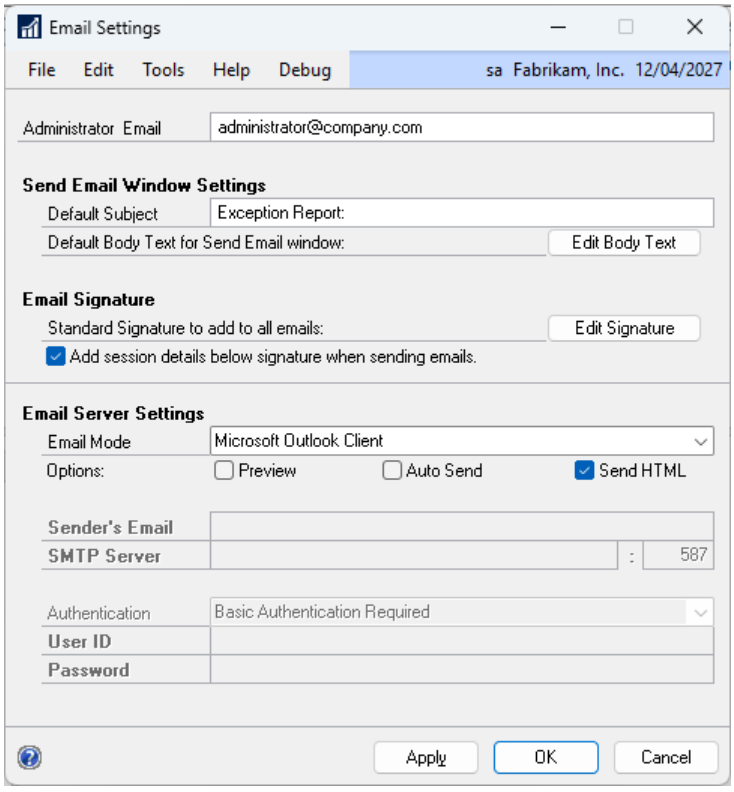
*Remove SQL Profile Trace SQL Components*

This button can be used to manually remove the SQL Profile Trace SQL Components (stored procedures) on the SQL Server.

## Email Settings

You can open the Email Settings window by selecting Email Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Email Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

The Email Settings window allows you define default values and server transport protocols and security settings to be used when sending emails from GP Power Tools.




The following is a description of the individual fields on the window:

*Administrator Email*

This field can be used to specify the default To email address(es) when sending emails.

Email addresses can be in the following formats and multiple addresses should be separated by a semi-colon (;):

- name@domain.com
- Full Name<name@domain.com>
- Full Name (when in Microsoft Outlook mode only)

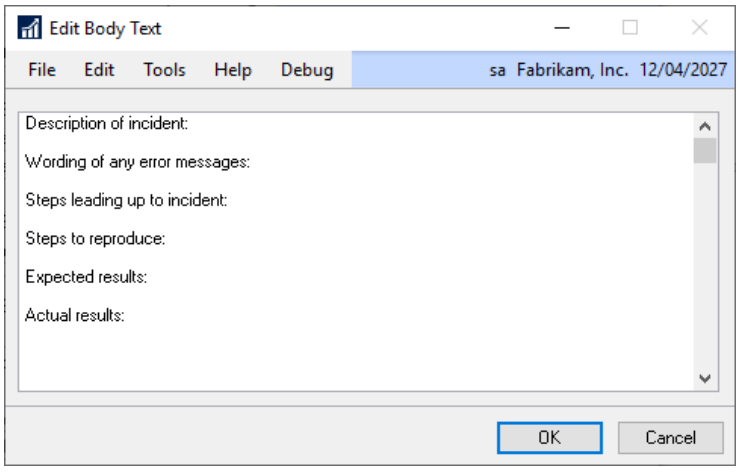
*Default Subject*

This field can be used to specify the default Subject line for the Send Email window.

*Default Body Text for Send Email window*

This button can be used to specify the default Body Text line for the Send Email window. This can be used to create a standard template for reporting issues to the system administrator.

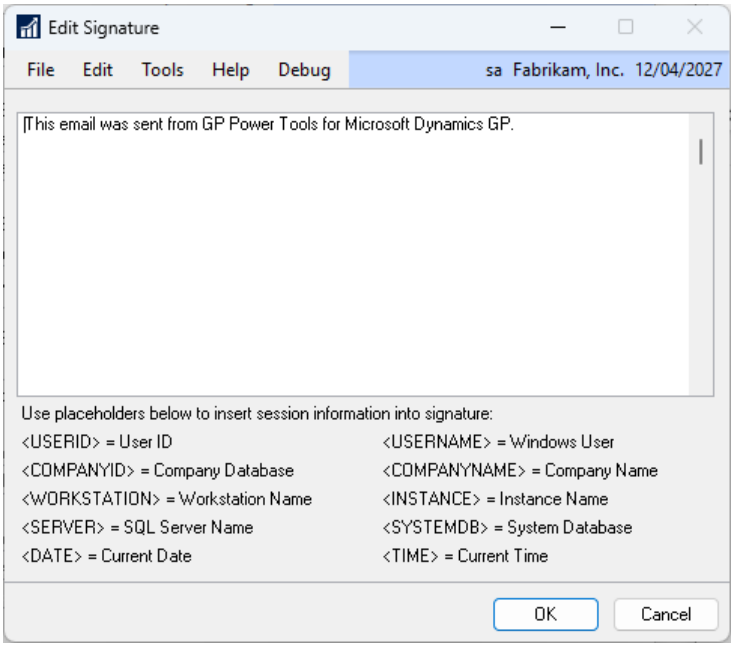
Click on the Edit Body Text Button to open the Edit Body Text window.




*Standard Signature to add to all emails*

This button can be used to create a standard signature to add to the bottom of all emails sent from GP Power Tools. If no signature is defined, the text in the screenshot below will be used.

Click on the Edit Signature Button to open the Edit Signature window.




*The signature can be customized with placeholders to insert information about the current session into the email when it is sent. You can also use the checkbox below to add session details automatically.*

*Add session details below signature when sending emails*

This checkbox can be used to automatically include details of the current session of Microsoft Dynamics GP below the signature when sending emails.

*Email Mode*

This field can be used to select whether the email engine is using a Microsoft Outlook Client (default setting) or a SMTP Server via CDO (Collaboration Data Objects) to send emails. Using SMTP instead of Outlook is useful for a Terminal Server environment where it is unlikely that an Outlook client is installed and set up on the Terminal Server.

You can also select to use any MAPI Compliant Client for sending emails. This will work for email clients other than Outlook if they are MAPI compliant. As Outlook is MAPI compliant, this mode also works for Outlook.

You can also select to use Exchange Web Services for sending emails. This will work directly with the Exchange Server and so does not require an email client to be installed.

For version 18.3 (for Microsoft Dynamics GP) and later, you can also select to use Multi-Factor Authentication for sending emails. This will use the MFA functionality added to Microsoft Dynamics GP to send emails.

Build 29 of GP Power Tools also adds SMTP Server via .Net Addin mode which supports the TLS protocol required to connect to an Office 365 Exchange Server via SMTP (smtp.office365.com:587).

*When running on the Web Client, the Microsoft Outlook Client email mode is not supported. It is recommended to use the SMTP or Exchange modes which do not require an email client.*

*Preview*

This option controls if the Send Email window is displayed whenever an email is sent.

*Auto Send*

This option controls if the email is automatically sent when an email is submitted. If Preview is unchecked, the email is submitted immediately, or if Preview is selected the email is submitted when the Send Button is clicked.

When using SMTP mode, Auto Send is always enabled. When using Outlook mode, this option controls whether the email is shown in the Outlook client before it is sent, without Auto Send the user will need to click the Send button in Outlook.

*Send HTML*

This option controls whether emails generated in GP Power Tools are sent as plain text or as HTML.

*Sender's Email*

This field must contain a single valid email address for use as the sender's email address when in SMTP mode. It is recommended to create a new email address for emails sent from Microsoft Dynamics GP.

The email address can be in the following formats:

- name@domain.com
- Full Name<name@domain.com>

*SMTP Server*

This field defines the SMTP Server's address. It can be specified as a name or as an IP address.

*SMTP Server Port*

This field defines the SMTP Server Port to use, the default value is 25.

*Authentication*

This drop-down list specifies what level of authentication is required to send emails via the SMTP Server. The options are:

- No Authentication Required
- Basic Authentication Required
- NTLM Authentication Required
- Basic Authentication & SSL Required
- NTLM Authentication & SSL Required

You can specify whether Basic or NTLM (Windows NT LAN Manager) Authentication is to be used and whether SSL (Secure Sockets Layer) should be used.

*User ID*

This field contains the user ID to login into the SMTP Server with. This would normally be the user ID associated with the Sender's Email defined above.

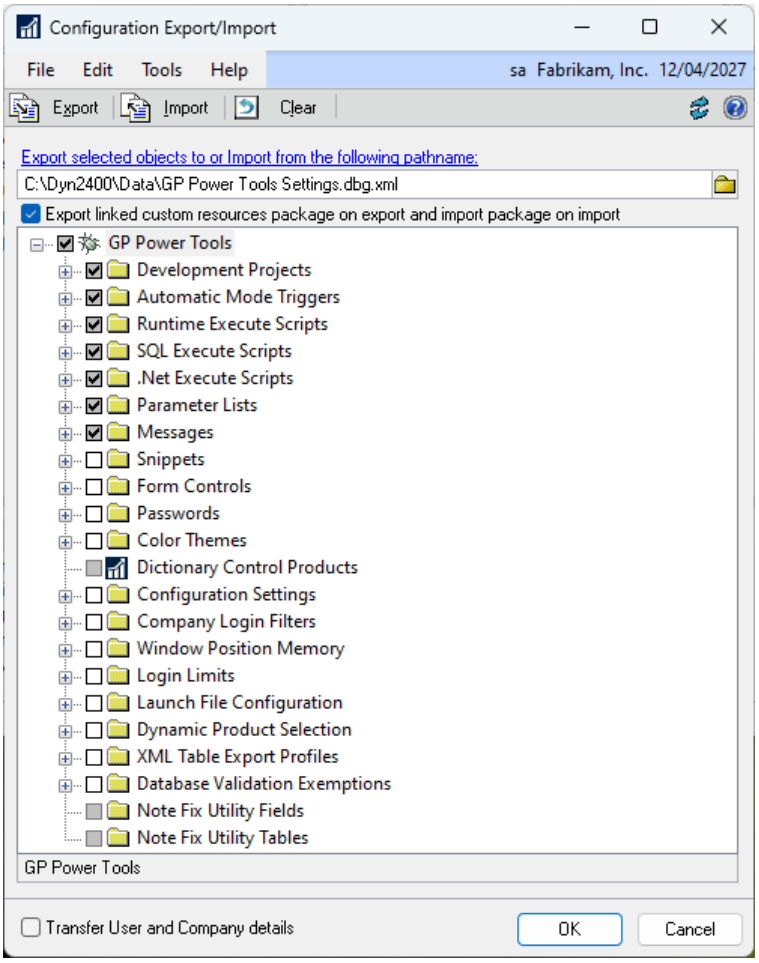
*Password*

This field contains the password to login into the SMTP Server with. This would normally be the password associated with the Sender's Email defined above.

## Configuration Export/Import

You can open the Configuration Export/Import window by selecting Configuration Export/Import from the Routines section of the GP Power Tools Area Page or by selecting Maintenance >> Configuration Export/Import from the Options button drop list on the main window. This is an Advanced Mode feature.

The Configuration Export/Import window can be used to export and import selected GP Power Tools settings.



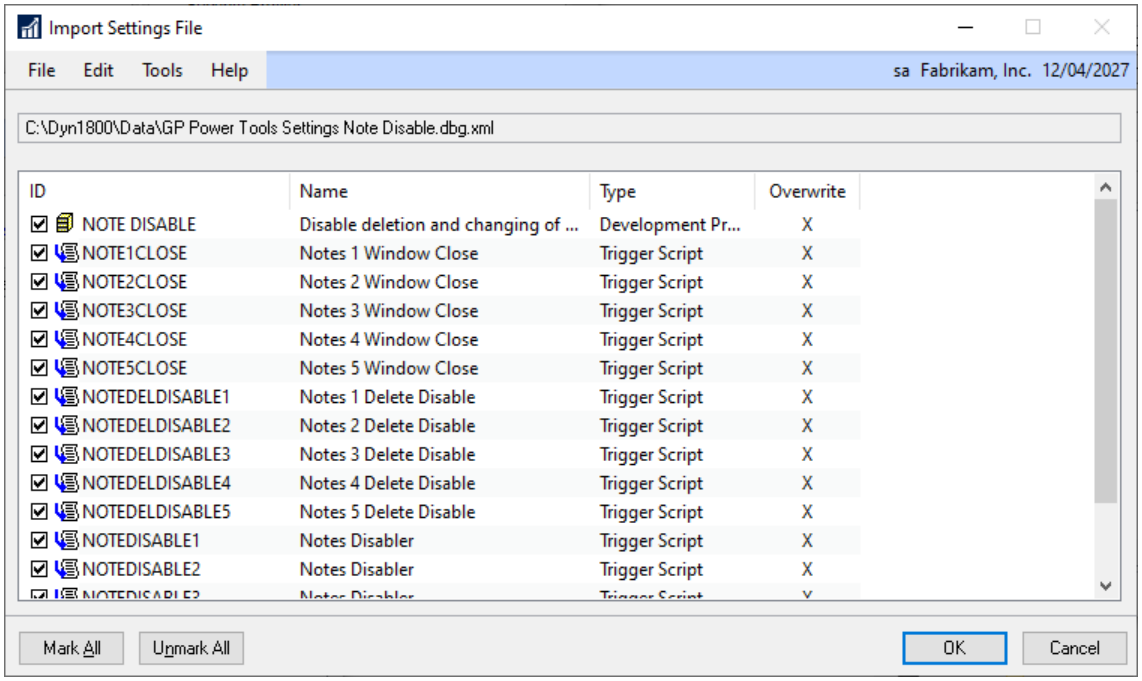

Below is a description of the individual fields on the window:

*Export Button*

This button will export the selected settings to the file name selected.

*Import Button*

This button will import the contents of the selected file name. It will open the Import Settings File window to display the contents of the settings file. You can then select the objects that will be imported from the settings file.




*Clear Button*

This button will clear any selections and reset the File Name and Transfer User and Company Details with Triggers checkbox.

*File Name*

This is the file name used for exporting and importing. The file should use the extension .dbg.xml.

*Export linked custom resources package on export and import package on import*

This checkbox enables exporting and importing of custom resources from Customization Maintenance along with the GP Power Tools resources.

*Transfer User and Company details*

This checkbox selects whether the user and company selection for triggers and products is exported when the trigger or Dictionary Control product is exported.

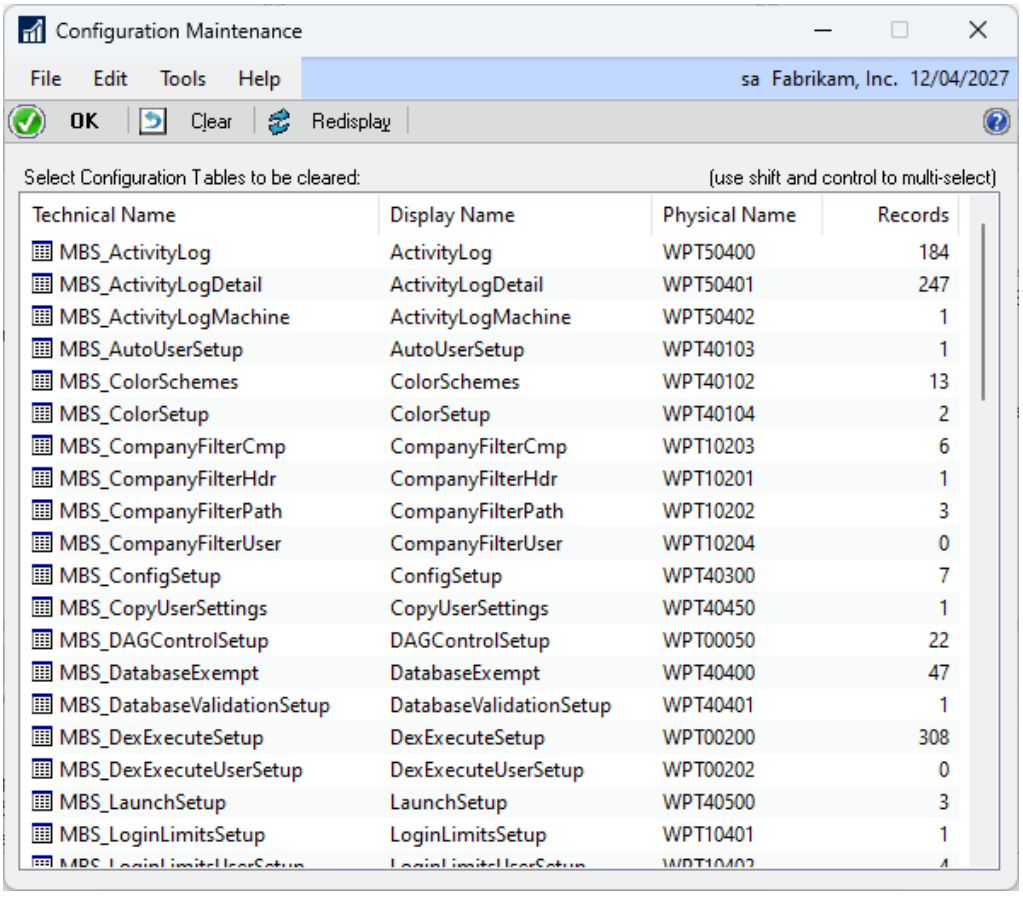
*If you select a Development Project from the tree, all triggers, scripts and parameter lists assigned to that project will automatically be selected. If you select a trigger, script or parameter list which belongs to a project, that project will be selected, but no other components will be selected. If you do not want to export the project, you can unselect it.*

*If you want to export all components of a Project, use the Project Setup window.*

## Configuration Maintenance

You can open the Configuration Maintenance window by selecting Configuration Maintenance from the Routines section of the GP Power Tools Area Page or by selecting Maintenance >> Configuration Maintenance from the Options button drop list on the main window. This is an Advanced Mode feature.

The Configuration Maintenance window can be used to clear the contents of the GP Power Tools settings tables. Click on the column headings to re-sort the list of tables.




The following is a description of the individual fields on the window:

*Clear Button*

This button will clear the contents of the selected tables. You can use the shift and control keys to select multiple tables.

*Redisplay Button*

This button will refresh window and update the record count.

*The system will always have a trigger ID named DEFAULT. This trigger will be automatically added when the MBS_TriggerSetup table is cleared.*

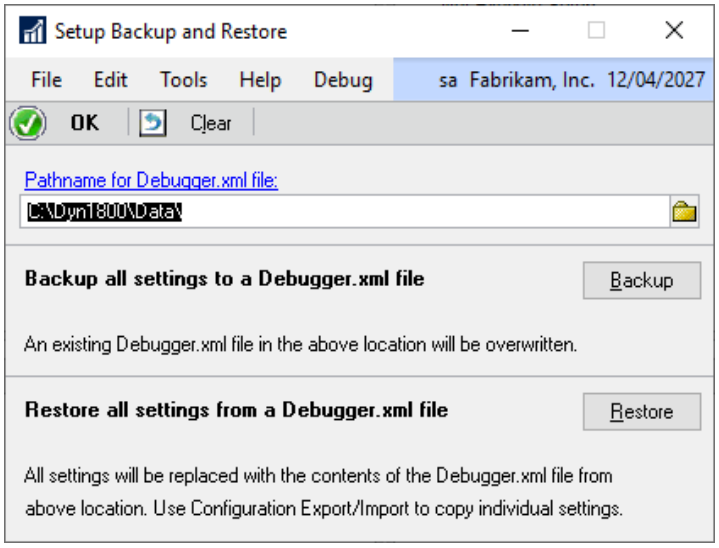
*To reset the Security Activity Tracking data used by the Security Log window, clear the contents of the MBS_SecurityLog and MBS_SecurityLogDetail tables.*

*To reset the User Activity Tracking data used by the User Activity Log window, clear the contents of the MBS_ActivityLog, MBS_ActivityLogDetail and MBS_ActivityLogMachine tables.*

## Setup Backup and Restore

You can open the Setup Backup and Restore window by selecting Setup Backup and Restore from the Routines section of the GP Power Tools Area Page or by selecting Maintenance >> Setup Backup and Restore from the Options button drop list on the main window. This is an Advanced Mode feature.

The Setup Backup and Restore window can be used to re-import a Debugger.xml file. It can also be used to backup and restore all settings from GP Power Tools.




The following is a description of the individual fields on the window:

*Pathname for Debugger.xml file*

This list contains the Dex.ini settings to be checked on login. The setting can be specified with an exact value (this is needed to add a new setting.

*Backup Button*

This Button will back up all settings to a file called Debugger.xml in the folder specified.

*Restore Button*

This will read the Debugger.xml file from the specified folder and replace all the settings from the imported file.

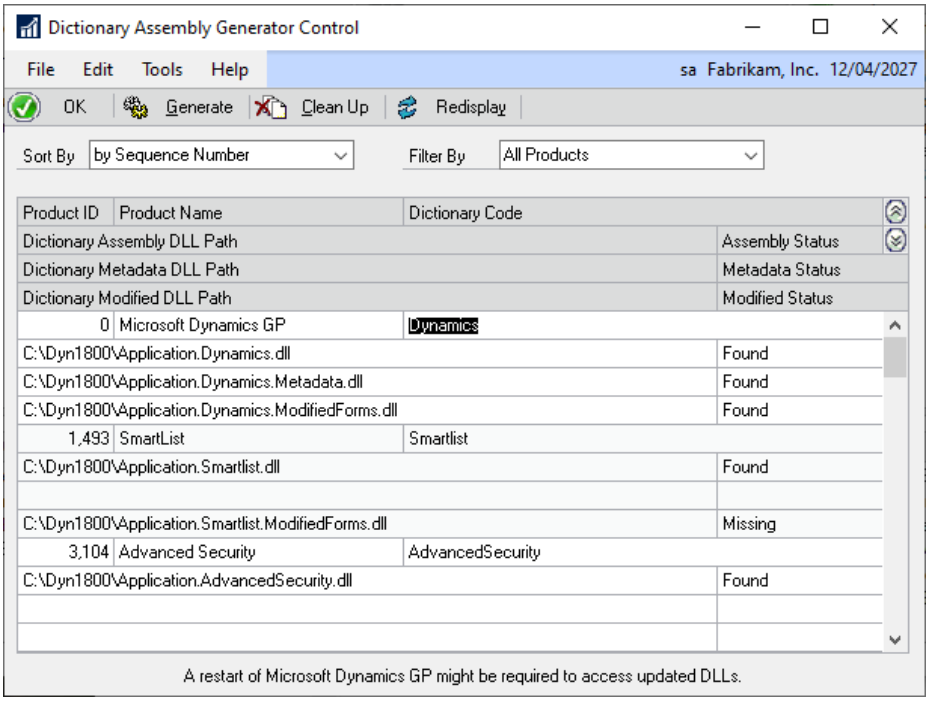
*You could use this window to reimport a Debugger.xml file if the file imported during the upgrade from a previous install was not the correct file. It can also be used to keep a backup of all settings. If you want to export and import individual settings, use the Configuration Export/Import window.*

*This window exports and imports all settings for GP Power Tools. It completely overwrites the target contents in the Debugger.xml file (for Backup) and the GP Power Tools SQL Tables (for Restore). Use with caution.*

## Dictionary Assembly Generator Control

You can open the Dictionary Assembly Generator Control window by selecting Dictionary Assembly Generator Control from the Routines section of the GP Power Tools Area Page or by selecting Maintenance >> Dictionary Assembly Generator Control from the Options button drop list on the main window. This is an Advanced Mode feature.

The Dictionary Assembly Generator Control window can be used to check for the existence of product dictionary assembly DLL files and create them if needed. It uses the DAG.EXE tool which is installed with GP Power Tools.




The following is a description of the individual fields on the window:

*Dictionary Code*

This field contains the default product dictionary identifier used when creating the pathnames for the associated Dictionary Assembly DLL files. It can be changed if desired when a product is not using the default name based on the product name listed in the Dynamics.set launch file.

*OK Button*

This button will close the Dictionary Assembly Generator Control window.

*Generate Button*

Use this button to generate the Application Dictionary Assembly, the Metadata Assembly (if the product contains Service Based Architecture (SBA) service procedures, and the Modified Forms Dictionary Assembly (if the Modifier has been used for the product).

*Clean Up Button*

Use this button to search for and remove renamed backup copies of Dictionary Assembly DLL files created when using the Generate Button. The backups are created as the DLL files might still be in use and can be renamed but not deleted.

*Redisplay Button*

This button will refresh the contents displayed on the Dictionary Assembly Generator Control window.

The Dictionary Assembly Generator Control window can also be opened from the .Net Execute Setup References window and from the Resource Information Script Parameters window.

*The Dictionary Assembly Generator Control window runs the DAG.EXE tool to create Dictionary Assembly DLL files based on the installed Product Dictionaries It renames any existing DLLs as they might be in use and cannot be deleted. You can use the Clean Up button after an application restart to remove the renamed files.*

*After using this window to recreate Assembly DLLs, you will probably need to restart Microsoft Dynamics GP to use the newly created DLL files.*

## Additional System Features

GP Power Tools adds some extra features to help users. Below is a summary of the features:

### Login Remember User

GP Power Tools fixes an issue where the Remember User feature on the login window does not work when user level Dex.ini files are being used. It makes sure that the RememberUser Dex.ini Setting is stored in the correct location.

### Remember Last Company

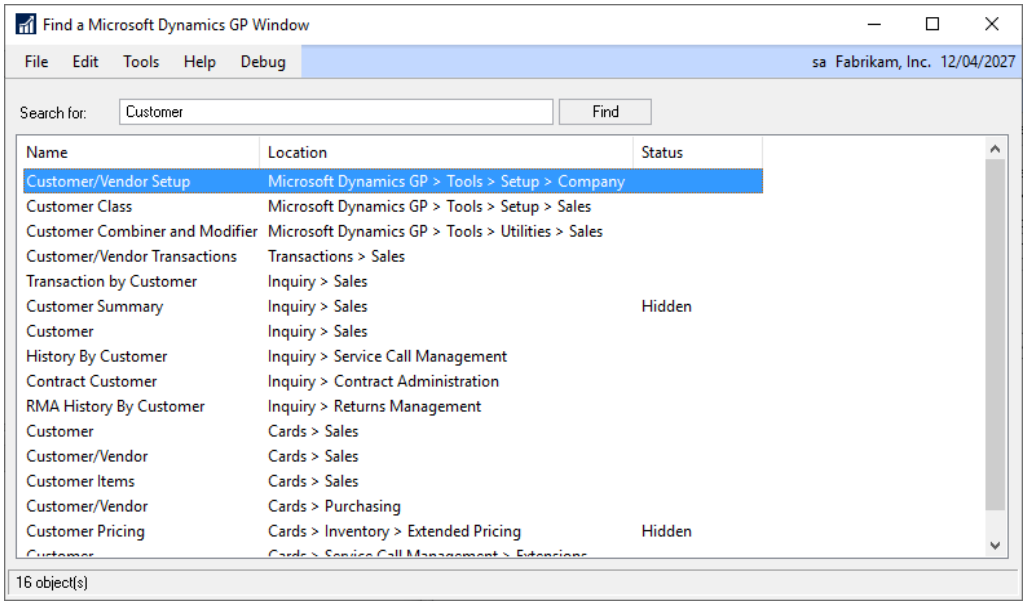
GP Power Tools remembers the last company logged into and selects that company when the Company Selection window is opened. The information is stored in the SQLLastCompany Dex.ini Setting.

### User Preferences Apply

GP Power Tools fixes the User Preferences window to that is the Apply button is used more than once without closing the window, it now works.

### Find a Window

GP Power Tools makes the Find a Microsoft Dynamics GP window feature available (just press Ctrl-F). It will search all menu navigation options for the specified text.




The Find a Window feature is also available from the Standard Toolbar.

## Raise All Windows

GP Power Tools adds the Raise All Windows menu option available from the application level menu and the Tools menu on all windows.

## Exit After Processes

GP Power Tools adds the Exit After Processes menu option available from the application level menu.

## Transaction being Edited

GP Power Tools adds the User Name to the "Transaction is being edited by another user" dialog. For Sales Order Processing: Sales Transaction Entry window and Purchase Order Processing: Purchase Requisition Entry window, Purchase Order Entry window, Purchasing Invoice Entry window and Receivings Transaction Entry window.


## Reload of User Dex.ini Settings

GP Power Tools reloads the User Dex.ini Settings after returning from Modifier or Report Writer because core Dynamics GP does not.

## Maintain Home Page Settings

GP Power Tools ensures that when the user's Home Page role is changed, the Home Page setting is maintained rather than defaulting back to Intelligent Cloud Insights.

## Mark User Inactive

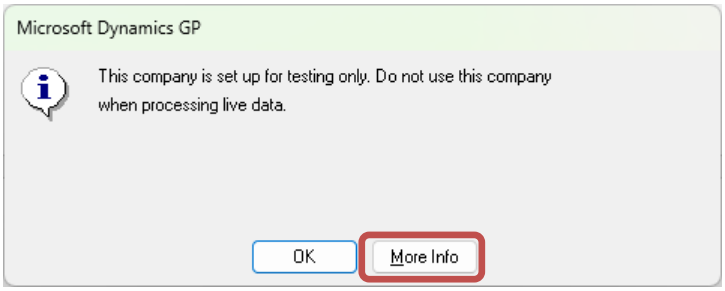
GP Power Tools ensures The Date Inactivated field in the SY_Users_MSTR (SY01400) table is updated when a user is marked as inactive.

## Multilingual Support for Test and Historical companies

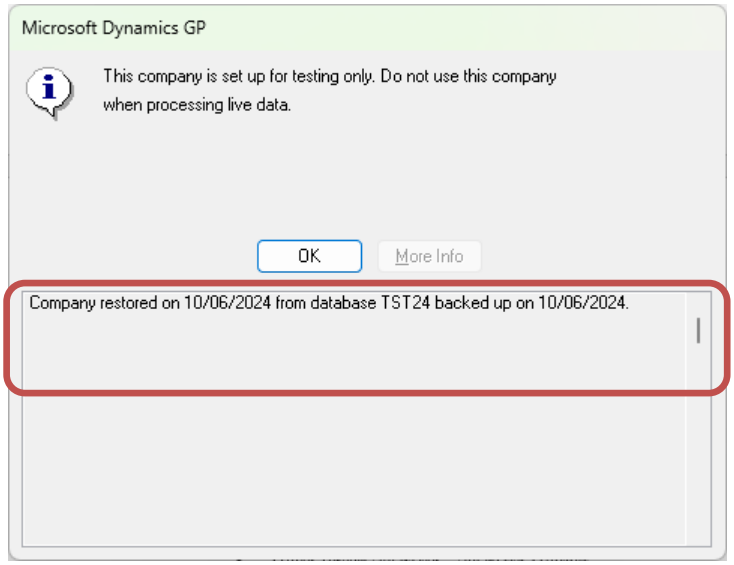
GP Power Tools resolves the issue when companies marked as Test or Historical with their Company Name containing <TEST> or <HISTORICAL> do not display the appropriate warning dialog on login on a non-English system.

## Database Information for Test and Historical companies

GP Power Tools also adds the ability to see when a Test and/or Historical company was last restored and the range of dates for transactions in the General Ledger.




Clicking the More Info Button will display the Database Information.



# Chapter 4: Administrator Tools Features

This chapter includes the following sections:

- *Resource Information*
- *Security Profiler*
- *Security Information*
- *Security Log*
- *Security Analyzer*
- *Enhanced Security*
- *Deny Based Security - Security Denied*
- *Deny Based Security - Security Hidden*
- *Administrator Settings**
- *Dex.ini Configuration**
- *Dictionary Control**
- *Company Login Filter**
- *Window Position Memory**
- *User Activity Log**
- *Login Limits**
- *Launch File Configuration**
- *Dynamic Product Selection**
- *Website Settings*
- *Product Version Validation*
- *Additional Administrator Features*

** Advanced Mode Feature*

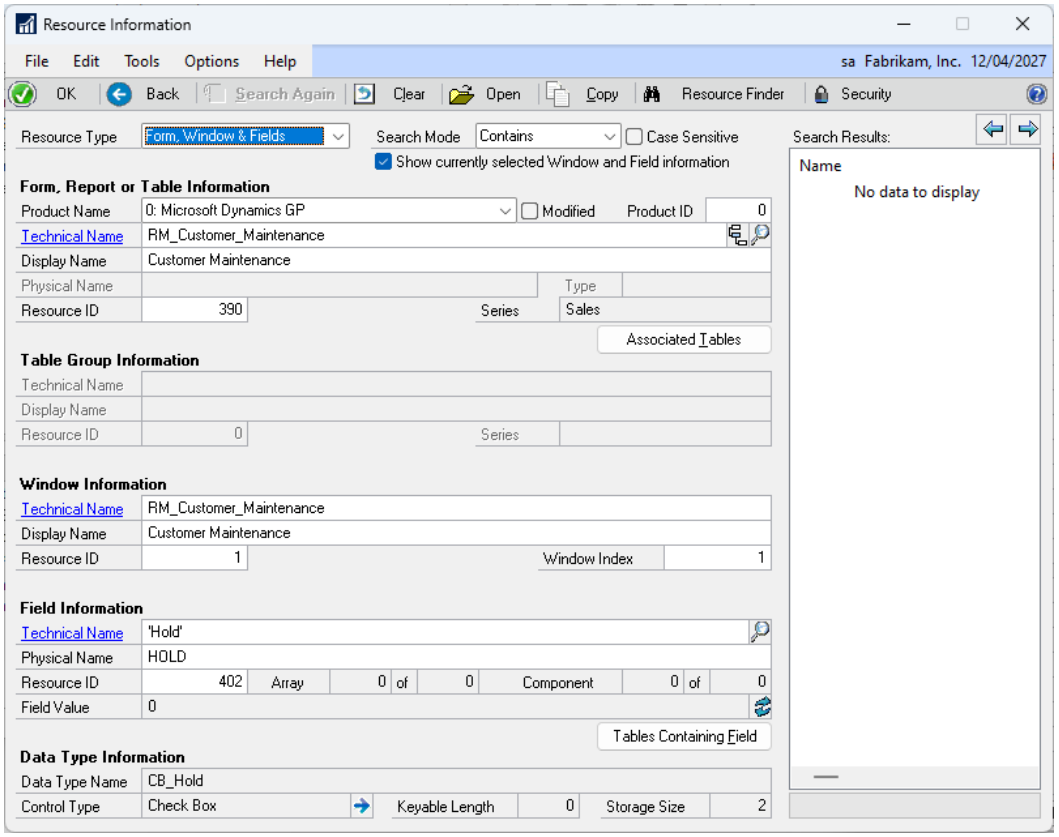
## Resource Information

You can open the Resource Information window by selecting Resource Information from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Resource Information from the Options button drop list on the main window.

*The Resource Information window can also be opened from the Tools menu or the Resource Descriptions menu on any window. When opened from these menus, the currently selected window field is automatically displayed in the window.*

The Resource Information window will display technical, display, and physical names and resource IDs for any form, window, field, table, table group, report, script (procedure or function, global or form level) global variable, constant, message or warning resource in the any dictionary currently installed in the Microsoft Dynamics GP application.

It can also provide information about non-dictionary resource Security Objects, such as Customization Tools, Document Access, Letters, Microsoft Dynamics GP Import, Navigation Lists, Series Posting Permissions, and SmartList Objects. If the products are installed, the following objects are also supported, SmartList Builder Permissions and Extender Resources. Security objects from other 3rd party products will show as Unknown Objects.




To use this window, enter the information you know into the appropriate field and the rest of the fields will be populated with the details for that resource.

For example, entering a window's display name will identify the window's technical name and resource ID; or entering a table's physical name as it appears in SQL Server will identify the table's dictionary, technical and display names as well as the resource ID.

*This window can be useful when working with table and column names in SQL Server, because it will quickly convert the physical names used in SQL back to the technical names used in Modifier, Report Writer and Dexterity.*

*For a field on a window on a form, if the form is open, the value of the field will be displayed in the Field Information section.*

Below is a description of the individual fields on the window:

### *OK Button*

This button will close the Resource Information window.

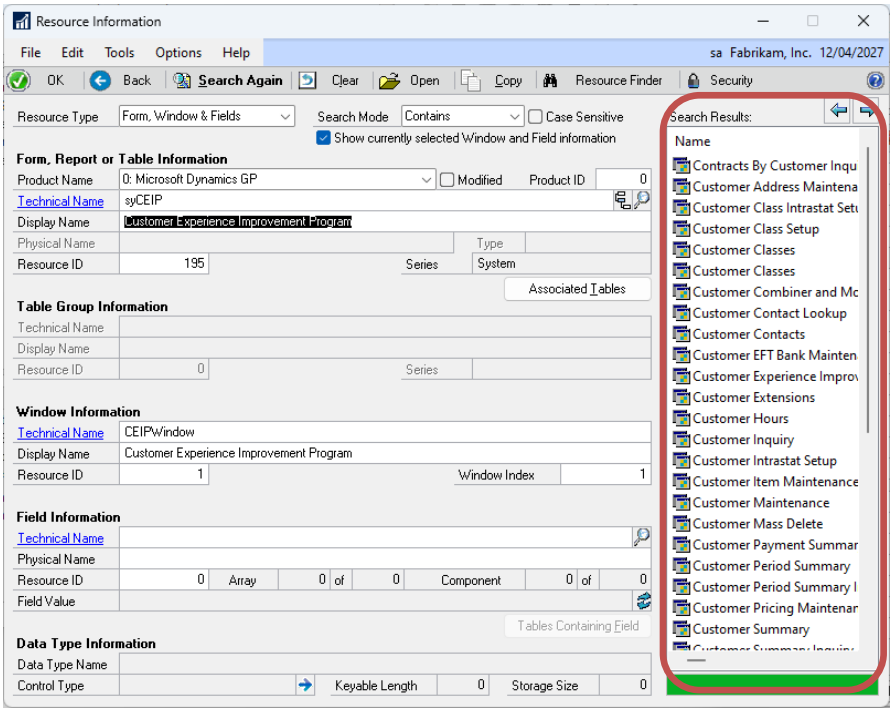
### *Back Button*

This button work backwards through the history of searched resources since the window was opened.

### *Search Again Button*

This button will search for the next resource to match the search criteria. Searching again works for Technical, Display and Physical Names for all resource types. The mode of the search can be controlled by the Search Mode drop-down list and the Case Sensitive checkbox.

You can also select from the Search Results list on the right-hand side of the window instead of using the Search Again button to scroll through the list individually.



*Clear Button*

This button will clear the current search in preparation for a new search.

*Open Button*

This button will open the selected form or report resource.

*Reports opened in this way will not have any options or restrictions applied and might contain unpredictable results. If the report uses a temporary table, this table will contain no data. Opening forms and reports from this window is only for testing purposes.*

*Copy Button*

This button will copy the selected resource to the clipboard so it can be pasted into scripts. It also can populate the Trigger Setup window's resource tab with the currently displayed resource.

*Resource Finder Button*

This button will open the Resource Finder window.

*Security Button*

This button will open the Security Information window for the selected resource. See sections below for more information. The Security Button will only be available if the current user has security access to the security windows under Tools >> Setup >> System.

*Resource Type*

This drop-down list controls whether Resource Information window is searching for Forms, Windows & Fields; Tables & Fields; Reports; Security Objects; Procedures & Functions; Messages & Warnings; Global Variables; or Constants.

*Search Mode*

This drop-down list controls how text searches will be handled by the Resource Information window. The options are Exact Match, Begins with and Contains. The default setting is Exact Match.

*Case Sensitive*

This checkbox controls if the text searches on the Resource Information window will be case sensitive or not. The default setting is to be case sensitive.

*Show currently selected Window and Field information*

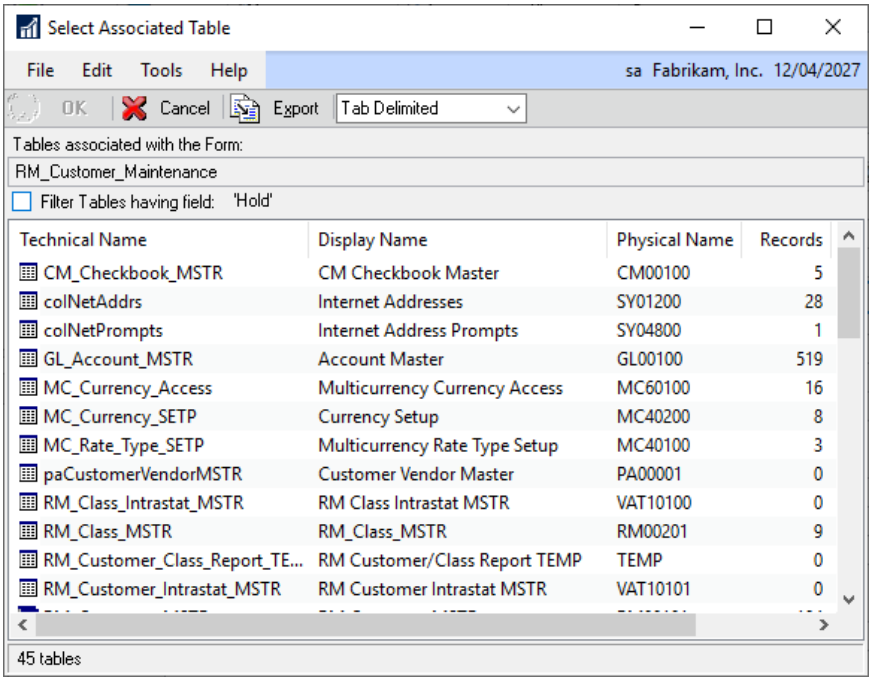
When this checkbox is selected for the Forms, Windows & Fields Resource Type, the Resource Information window will automatically display the details for the currently selected Form, Window and Field.

*The Show currently selected Window and Field information feature only works for windows opened while the Resource Information or Resource Finder windows are open. So open the Resource Information window before opening the windows you want information about.*

*Associated Tables Button*

This button is available when the Resource Information window is in Forms, Windows & Fields mode. It will display a list of tables associated with the currently selected form.

If a field is selected on the Resource Information window, you will have the option to filter the list of tables to only tables having the specified field. If the field is not available in any tables, this option will be disabled.

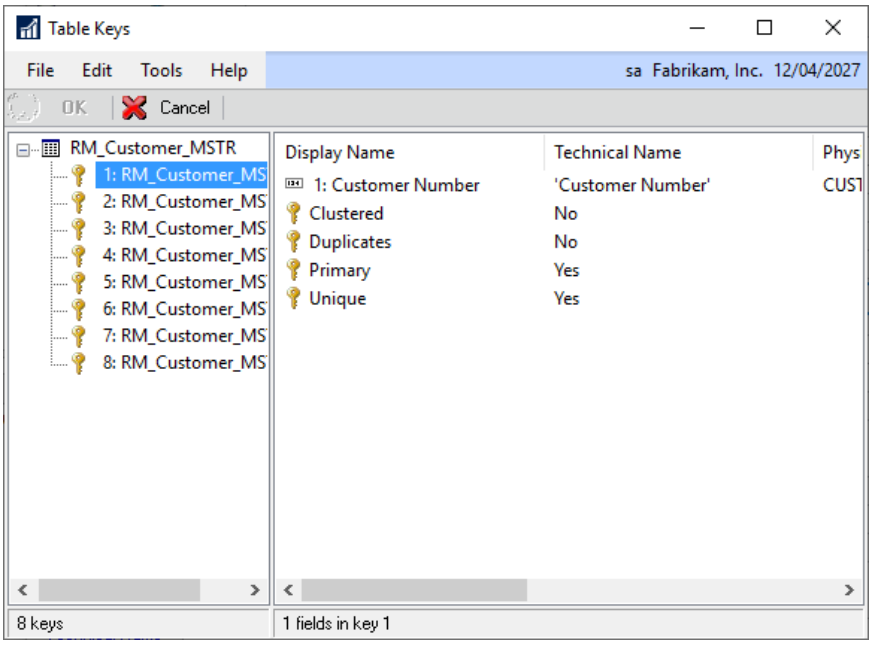



*This linked table for the window is highlighted with different icon and a flag in the Linked column in the display.*

Selecting a table from this window will change the Resource Information window into Table & Field mode and display the details of the selected table. If filtering on a field, the field will also be selected.

*Display Keys Button*

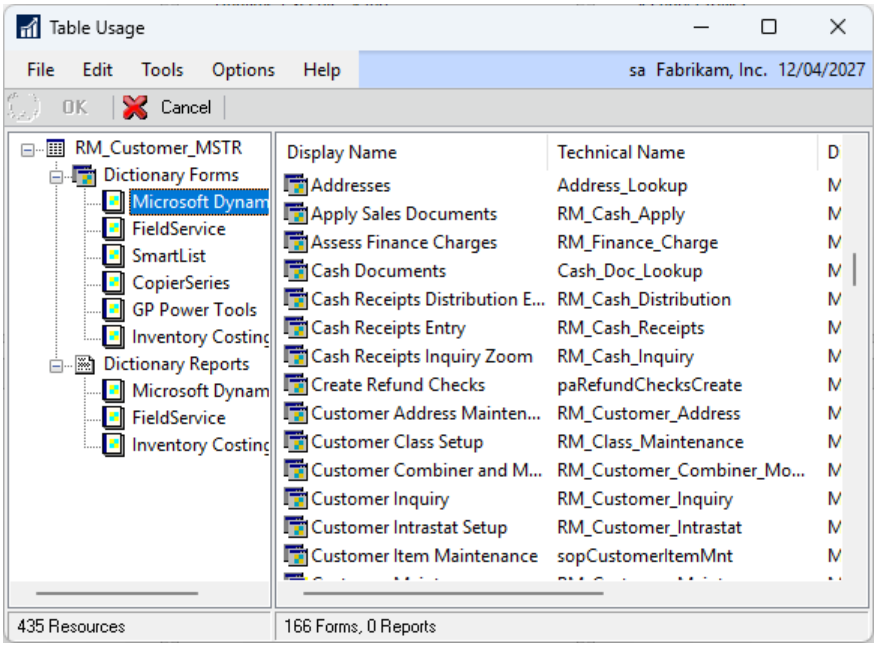
This button is available when the Resource Information window is in Tables & Fields modes. It will display a list of keys (indexes) for the currently selected table. The fields for the key and the key options are displayed.




Selecting a field from this window will display the details of the selected field.

### *Display Usage Button*

This button is available when the Resource Information window is in Tables & Fields modes. It will display a list of windows and reports which use the currently selected table.



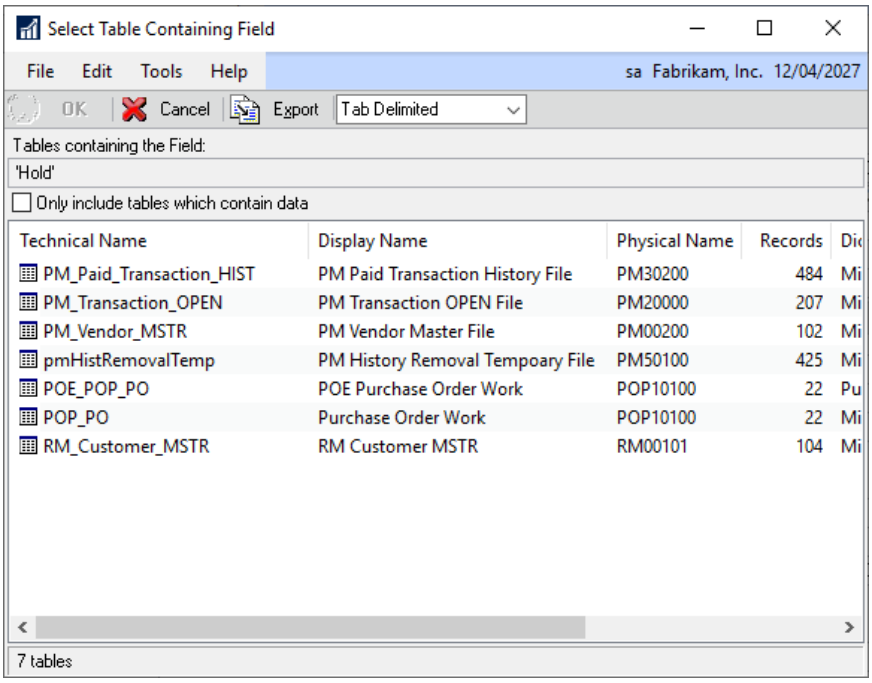

Selecting a window or report from this window will display the details of the selected resource.

### *Redisplay Field Button*

This button can be used to refresh the Field Value information displayed if the field value has changed.

### *Tables Containing Field Button*

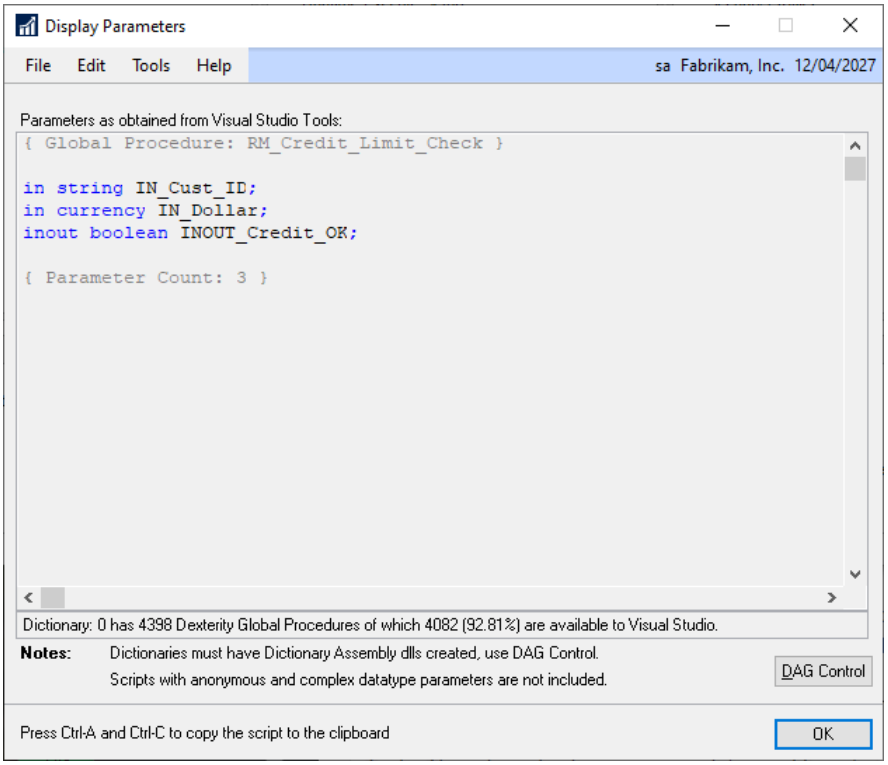
This button is available when the Resource Information window is in Forms, Windows & Fields and Tables & Fields modes. It will display a list of tables which contain the currently selected field. You have the option to select to only include tables which contain data.




Selecting a table from this window will change the Resource Information window into Table & Field mode and display the details of the selected table and field.

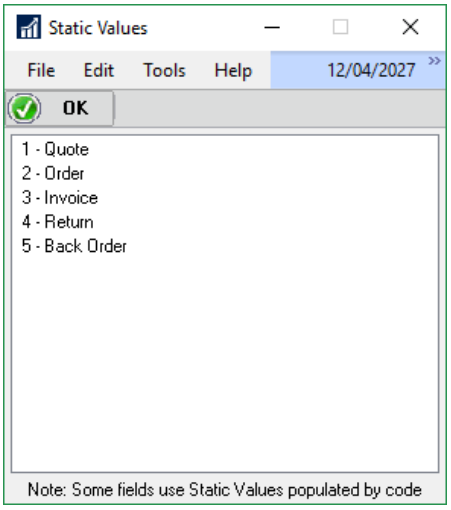
### *Display Parameters Button*

This button is available when the Resource Information window is in Procedures & Functions mode. It will attempt to display a list of parameters for the currently selected procedure or function.




*This functionality used Visual Studio Tools to read the Dexterity parameters from the Dictionary Assembly DLL files created for each dictionary. Not all procedures and functions are exposed to Visual Studio Tools, so scripts with anonymous or complex datatype parameters might not be found. If the Dictionary Assembly for a product dictionary is not available, click on the DAG Control Button to open the Dictionary Assembly Generator Control window which can be used to generate it.*

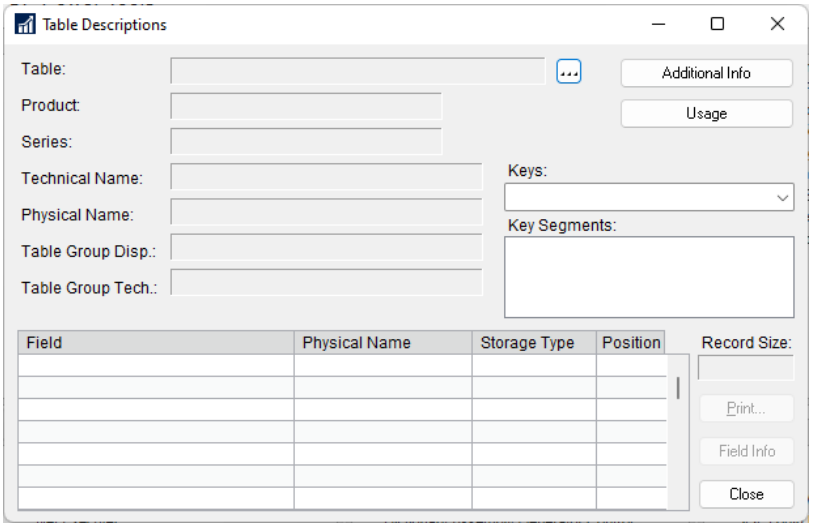
Next to the Control Type field is the Static Values expansion button which displays the Static Values associated with the data type.




The following is a description of the Options menu available for the window:

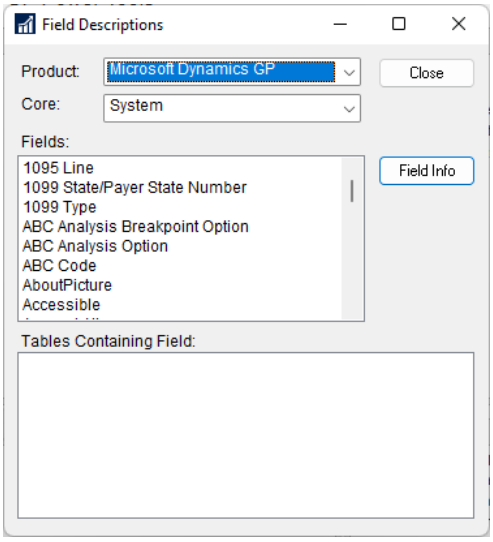
*Table Descriptions*

Use this menu option to open the default Table Descriptions window. This also allows this window to be opened on the Web Client.



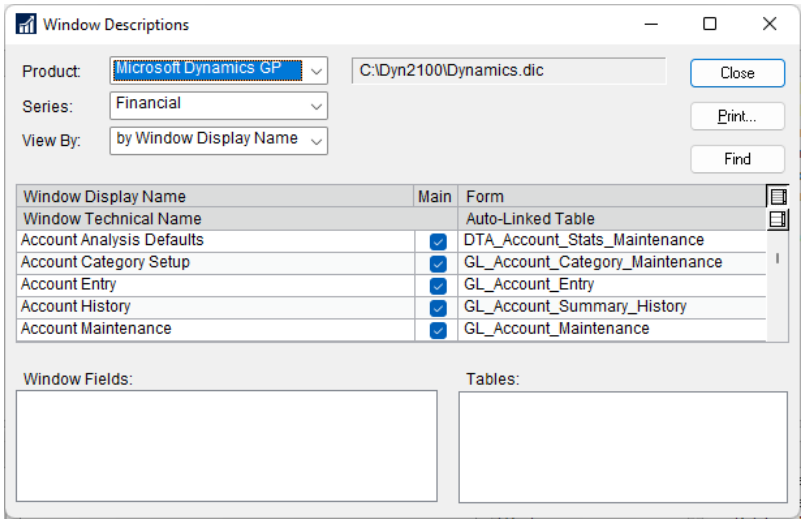
### *Field Descriptions*

Use this menu option to open the default Field Descriptions window. This also allows this window to be opened on the Web Client.



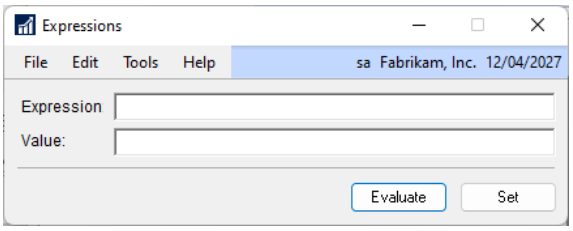

### *Window Descriptions*

Use this menu option to open the default Window Descriptions window. This also allows this window to be opened on the Web Client.



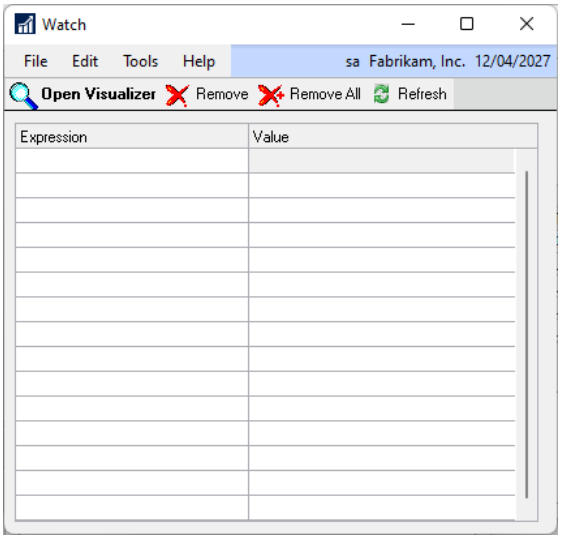
### *Debug Expressions*

Use this menu option to open the Script Debugger Expressions window. When the Expressions window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



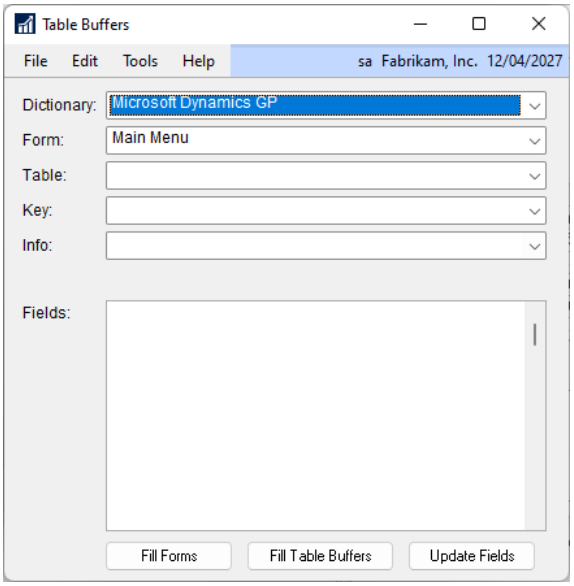

### *Debug Watch*

Use this menu option to open the Script Debugger Watch window. When the Watch window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



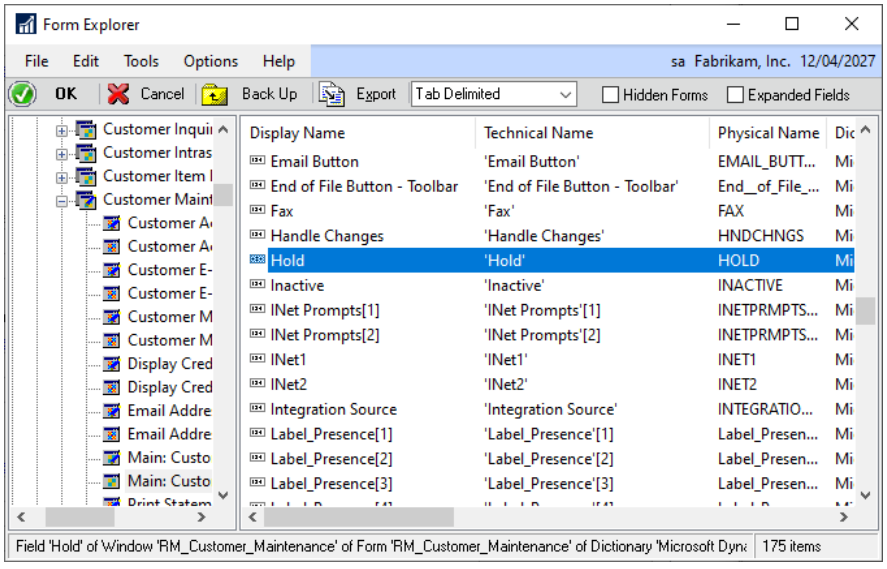
### *Debug Table Buffers*

Use this menu option to open the Script Debugger Table Buffers window.




When in Form, Window & Field mode, you can use the lookup button to select a form, window or field resource. Once clicked the Form Explorer window will open.

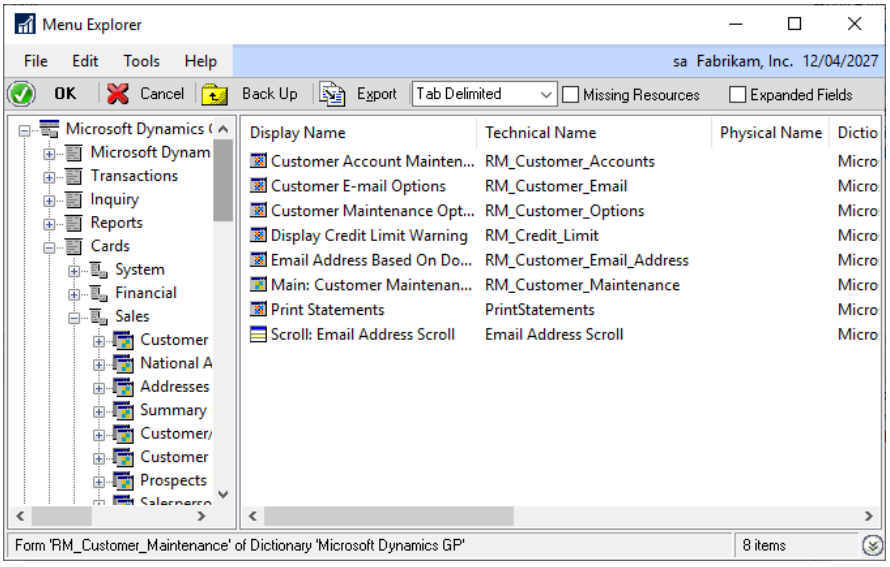
*The Form Explorer can show modified and alternate resources. Modified resources are shown with a blue pencil. Alternate resources are shown with a red pencil. When fields on a modified or alternate window are displayed, only fields not on the original window will be highlighted.*



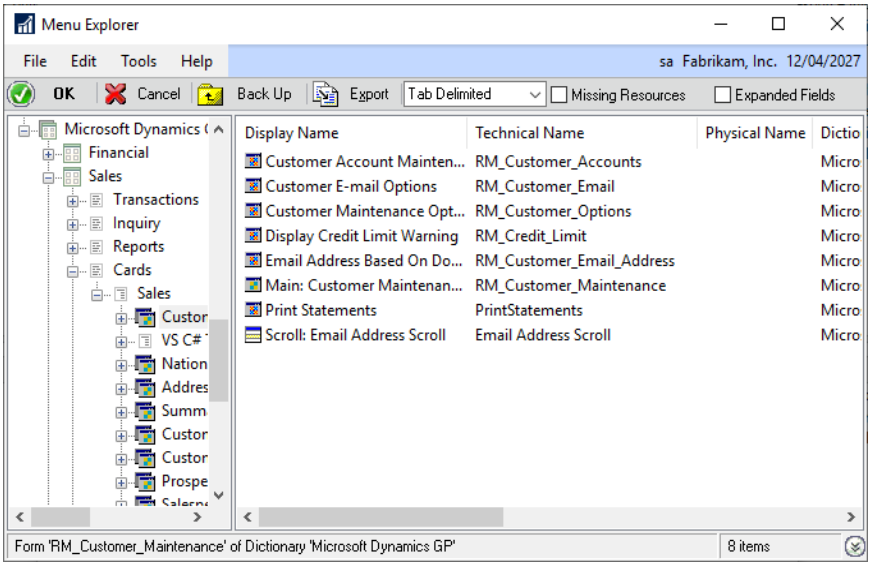
*To insert a form name or window name, select the resource in the right-hand pane and click OK. If no resources are selected on the right-hand pane, the currently selected resource in the left-hand pane will be used when OK is clicked. Clicking on the resource name in the status field at the bottom of the window toggles Dexterity and .Net view.*

You can also use the menu lookup button  to select a form, window or field resource based on the menu navigation model. Once clicked the Menu Explorer window will open.

*The Menu Explorer can show modified and alternate resources. Modified resources are shown with a blue pencil. Alternate resources are shown with a red pencil. When fields on a modified or alternate window are display, only fields not on the original window will be highlighted.*

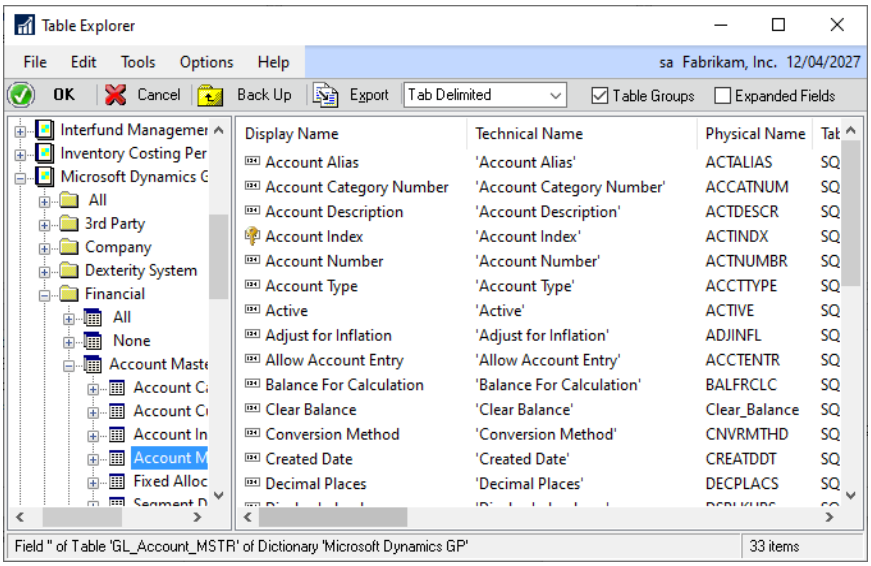



The Menu Explorer window has the option to navigate via application menus (top of left pane) or by the Area Pages (bottom of left pane).



*To insert a form name or window name, select the resource in the right-hand pane and click OK. If no resources are selected on the right-hand pane, the currently selected resource in the left-hand pane will be used when OK is clicked.*

When in Table & Field mode, you can use the lookup button to select a table or field resource. Once clicked the Table Explorer window will open.

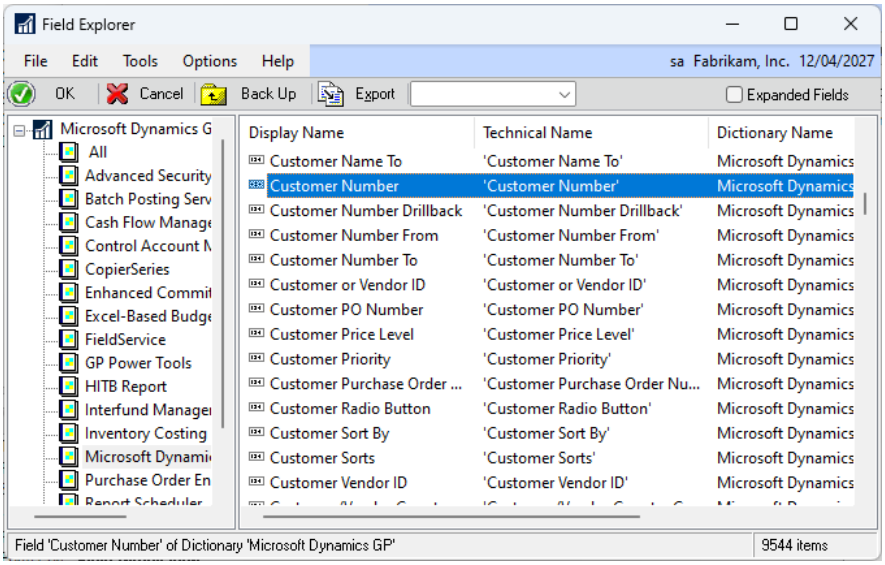



*To insert a table name, select the resource in the right-hand pane and click OK. If no resources are selected on the right-hand pane, the currently selected resource in the left-hand pane will be used when OK is clicked. Clicking on the resource name in the status field at the bottom of the window toggles Dexterity and .Net view.*

The Table Explorer window highlights the primary key fields, but can also display other key (index) information for a table. Expand the table node in the tree to display the keys; selecting an individual key will display the key fields and the key options.

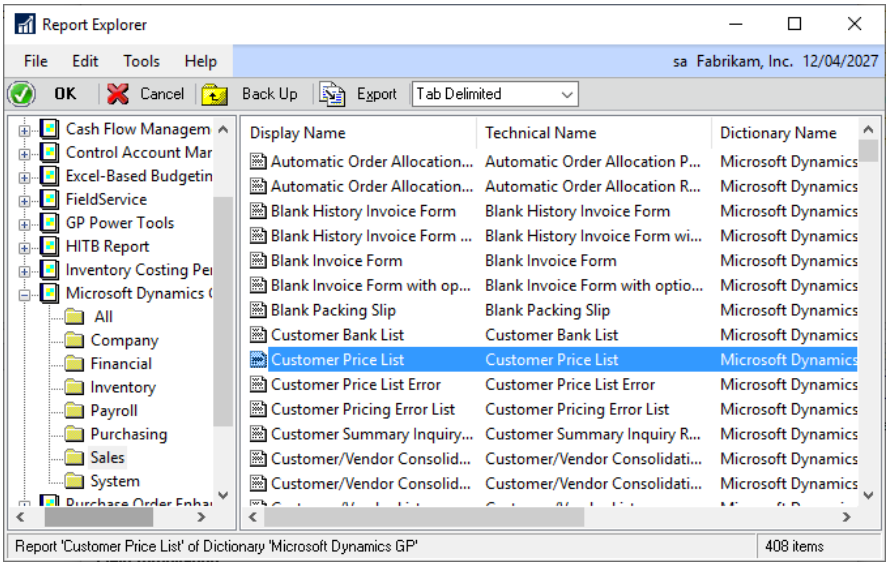
The Table Explorer window has the option to navigate to tables with or without table groups. Use the Table Groups checkbox to change views.

When in Form or Table modes and the Field information can be entered, you can use the lookup button next to the Field Technical Name to select a field resource. Once clicked the Field Explorer window will open.



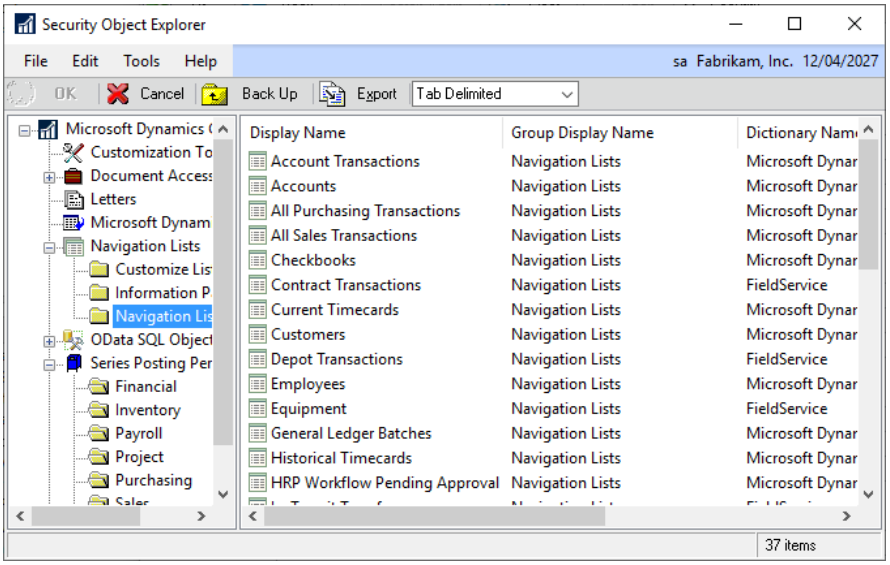

*To insert a field name, select the resource in the right-hand pane and click OK.*

When in Report mode, you can use the lookup button to select a report resource. Once clicked the Report Explorer window will open.



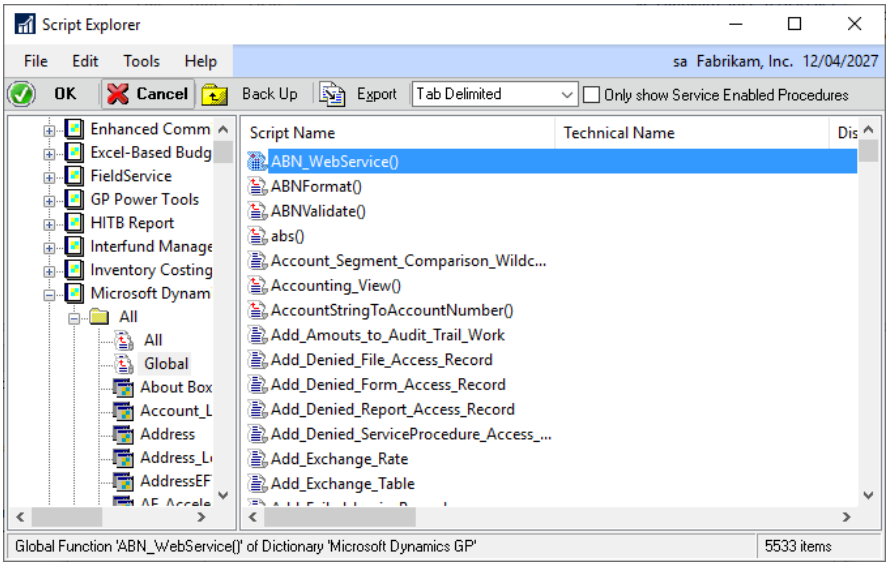

*To insert a report name, select the resource in the right-hand pane and click OK. Custom Reports are shown with a different icon in the right-hand pane.*

When in Security Object mode, you can use the lookup button to select a security object. Once clicked the Security Object Explorer window will open.



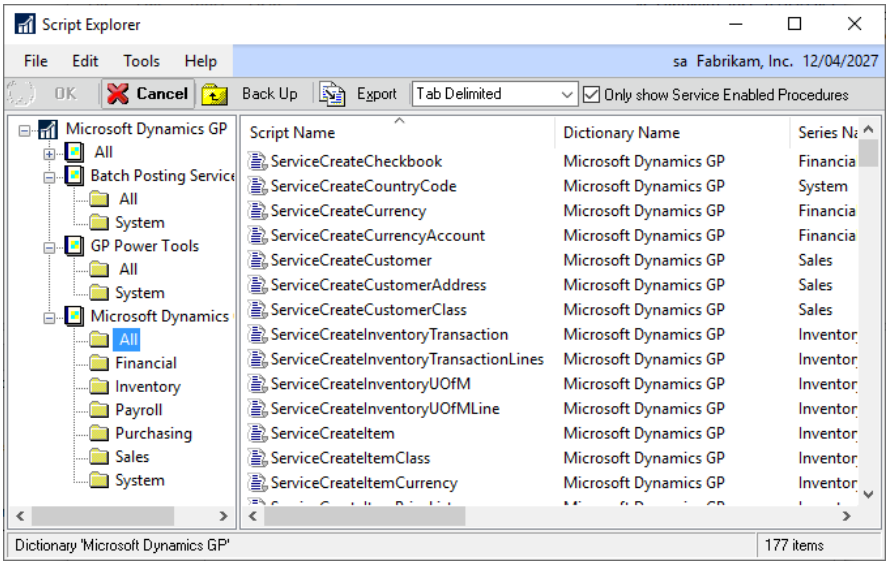

*To insert a security object, select the desired security object in the right-hand pane and click OK. Security objects from other 3rd party products will show as Unknown Objects.*

When in Procedure & Function mode, you can use the lookup button to select a script resource. Once clicked the Script Explorer window will open.

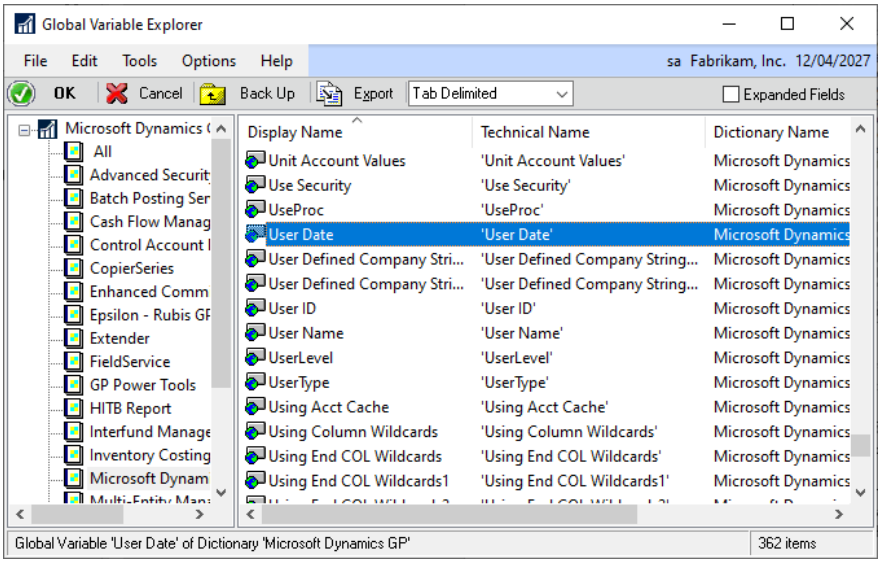



*To insert a script name, select the resource in the right-hand pane and click OK. Procedures and Functions are shown with different icons in the right-hand pane. Clicking on the script name in the status field at the bottom of the window toggles Dexterity and .Net view.*

You can select to only show Service Enabled Procedures, which shows a simplified tree structure in the left-hand pane.

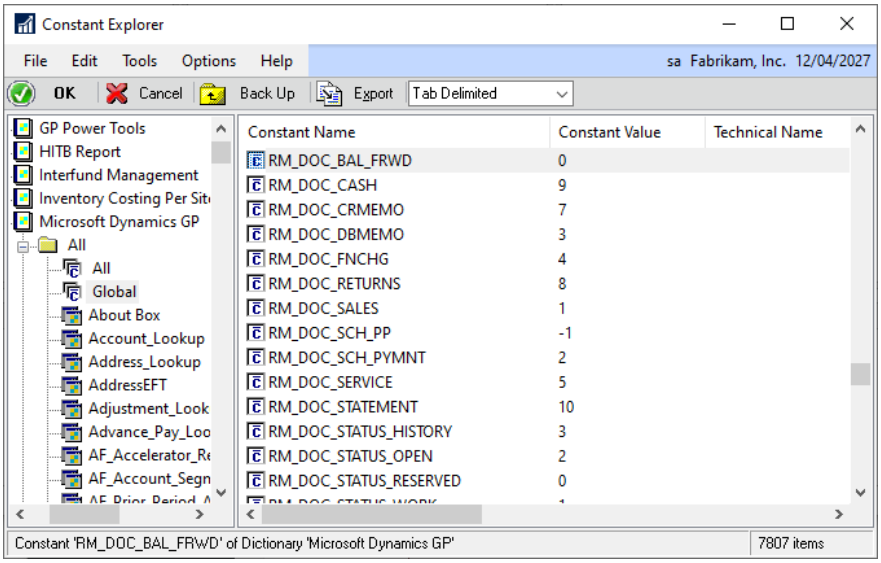



When in Global Variables mode, you can use the lookup button to select a global variable resource. Once clicked the Global Variable Explorer window will open.




*To insert a global variable name, select the resource in the right-hand pane and click OK.*

When in Constants mode, you can use the lookup button to select a constant resource. Once clicked the Constant Explorer window will open.




*To insert a constant or form constant name, select the resource in the right-hand pane and click OK.*

Below is a description of the individual fields on the Resource Explorer windows:

### *OK Button*

This button will return the selected resource and close the window.

### *Cancel Button*

This button will close the window without making a selection.

### *Back Up Button*

This button will change the current selection to its parent on the tree.

### *Export Button*

This button will allow the resources displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

### *Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

### *Hidden Forms*

Use this check box to show forms which are normally hidden from the security system.

### *Missing Resources*

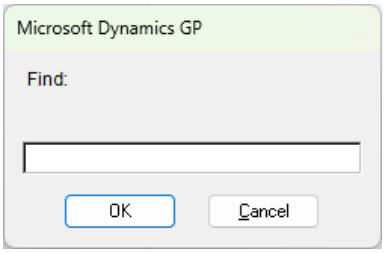
Use this check box to show menu items which point to external or missing resources.

*Expanded Fields*
Use this check box to expand composite and array fields into the component parts.

*Only show Service Enabled Procedures*
Use this check box to limit the Script Explorer to only show Service Enabled Procedures.

*Options Menu >> Redisplay*
Use this menu option to redisplay the contents of the window.

*Options Menu >> Find*
Use this menu option to open a dialog to select the text to search for.




*Options Menu >> Find Next*
Use this menu option to search for the previously entered text.

*Options Menu >> All Columns*
Use this menu option apply the search to all columns in the right-hand pane.

*Options Menu >> Sort Column*
Use this menu option apply the search to only the current sort column in the right-hand pane.

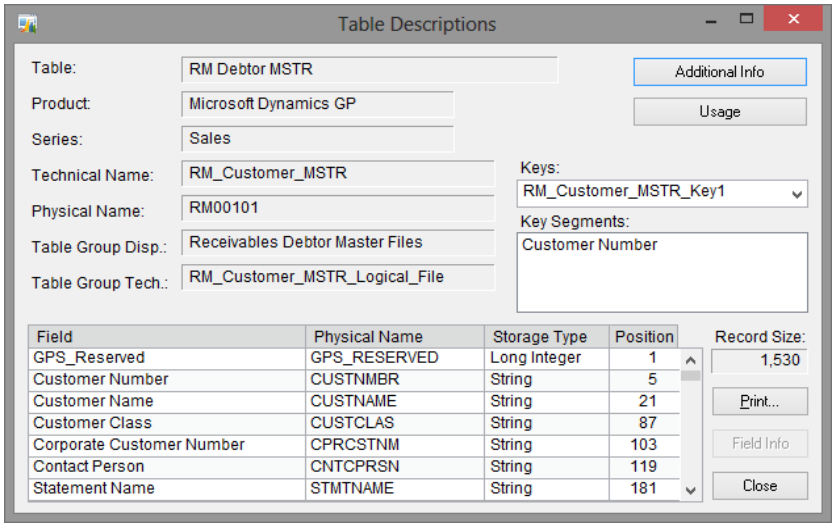
*Options Menu >> Contains*
Use this menu option select a contains search.

*Options Menu >> Begins With*
Use this menu option select a begins with search.

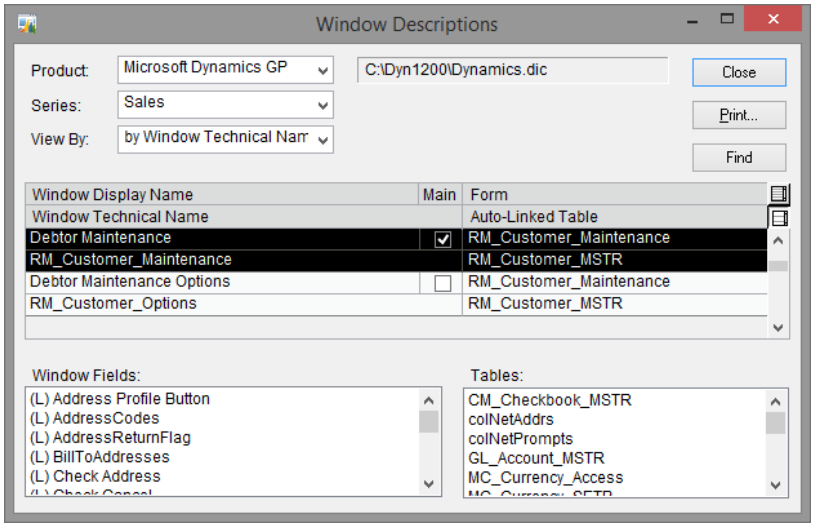
*The Resource Explorer windows which have two panes are Splitter enabled which allows the ratio between the left and right-hand panes to be adjusted. When running on the Web Client, the splitter functionality is disabled.*

*To improve overall performance, the Resource Explorer windows use SQL based cache tables. The tables are populated when the window is first opened and are automatically updated whenever a product dictionary is added or updated. To manually reset the cache tables and to re-read the product dictionaries, select Options >> Refresh Dictionary Resources from the window menu.*

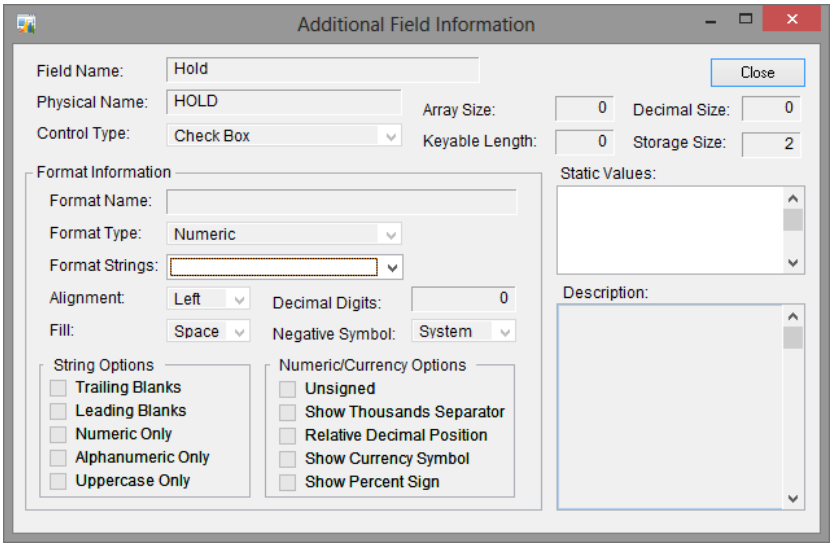
On the Resource Information window, you can click the Open Button or the Technical Name hyperlink to open the current resource. If the resource is a form or report, it will open. If the resource is a table, the standard Table Descriptions window will open.



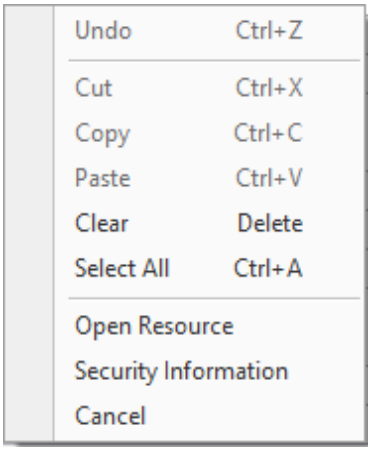

If you click Window Technical Name hyperlink, the standard Window Descriptions window will open.



If you click Field Technical Name hyperlink, the standard Field Information window will open.




*The Resource Information window is Right click enabled. If you right mouse click on any of the fields you can select Open Resource (same as Open Button), Security Info (same as Security Button) or Cancel from the context sensitive menu. The Security Info option will only be available if the current user has security access to the security windows under Tools >> Setup >> System.*

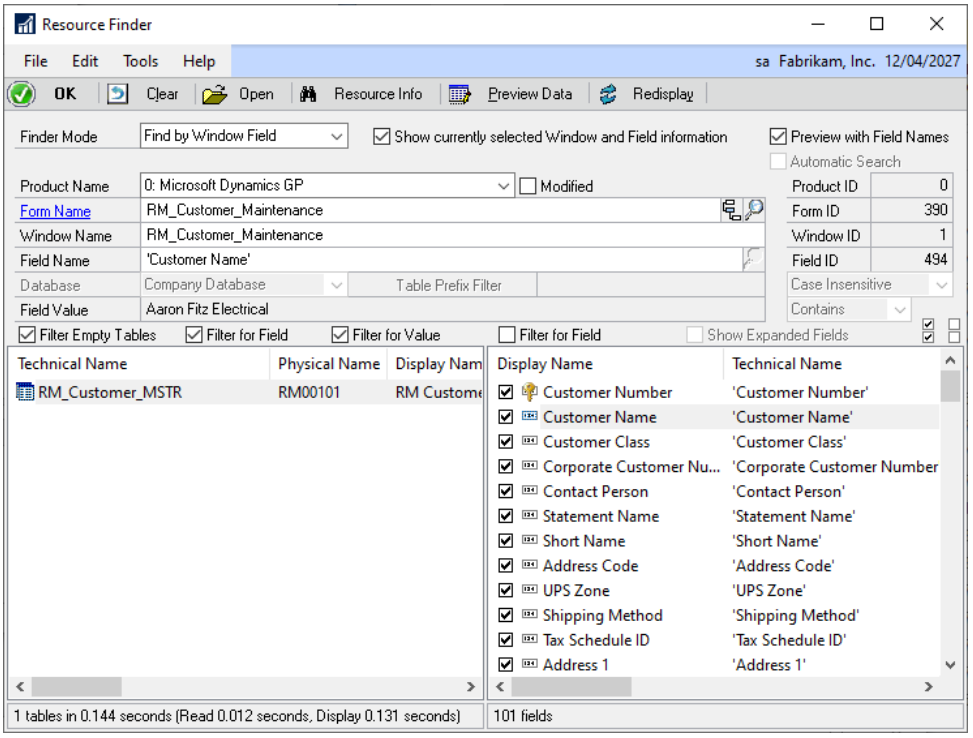


## Resource Finder

You can open the Resource Finder window by selecting Resource Finder from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Resource Finder from the Options button drop list on the main window.

*The Resource Finder window can also be opened from the Tools menu or the Resource Descriptions menu on any window. When opened from these menus, the currently selected window field is automatically displayed in the window.*

The Resource Finder window is designed to help identify exactly where data from a window field is stored in the application's tables. It combines and extends the functionality of the Resource Information window.




The window has three modes:

- Find by Window Field: It can find the table fields based on a window field by looking at the form's associated tables and filtering out empty tables, and including those tables that contain the field and also for those tables containing the field's value.

- Find by Table Field: It can locate all tables which contain the field and filter to exclude empty tables.

- Find by Field Data: It can scan an entire database for the specified field value and return the table and column of everywhere it is found. To improve performance of this search you can filter for tables starting with a prefix and/or for a specific Field (Column) Name.

The Product, Form, Window and Field names can be manually entered, selected using the Form Explorer or Menu Explorer windows. The Field Name can also be

Below is a description of the individual fields on the window:

*Filter Mode*

Use this field to select the desired filter mode as described above. Find by Window Field is the default mode.

*OK Button*

This button will close the Resource Finder window.

*Clear Button*

This button will clear the current search in preparation for a new search.

*Open Button*

This button will open the selected form resource.

*Resource Info Button*

This button will open the Resource Information window.

*Preview Data Button*

This button will open the SQL Execute Setup window to preview the data for the selected fields in the SQL table. The Preview with Field Names option controls if the Dexterity Technical Names or SQL Physical Names are used as the column headers.

*Previewing data uses the SQL Execute Setup window to display the data and so needs the Developer Tools module registered.*

*Redisplay Button*

This button will repeat the search and redisplay the results.

*Show currently selected Window and Field information*

When this checkbox is selected for the Forms, Windows & Fields Resource Type, the Resource Finder window will automatically display the details for the currently selected Form, Window and Field.

*The Show currently selected Window and Field information feature only works for windows opened while the Resource Finder or Resource Information windows are open. So, best practice is to open the Resource Finder window before opening the windows you want information about.*

*Preview with Field Names*

This checkbox controls if the Dexterity Technical Names or SQL Physical Names are used as the column headers when previewing data.

*Auto Search*

This checkbox is used for Find by Field Data mode to control whether to automatically search when individual settings are changed. When Auto Search is not selected, use the Redisplay Button to manually start the search.

*Filter Empty Tables*

This checkbox controls if empty tables should be excluded from the search results.

*Filter for Field*

This checkbox controls if the search results should be filtered to only include tables that contain the selected field.

*Filter for Value*

This checkbox controls if the search results should be filtered to only include tables that contain the specified data in the selected field.

*Filter for Field (Field List)*

This checkbox will only show the selected field in the Field List.

*Show Expanded Fields*

Use this check box to expand composite and array fields into the component parts.

*Case Mode*

This drop-down list is used for Find by Field Data mode to control whether the search should be Case Insensitive, Case Sensitive or use the Default Sensitivity for the SQL Server.

*Search Mode*

This drop-down list is used for Find by Field Data mode to control whether to use an Exact Match, Begins With or Contains search.

*Mark All*

Use this button to mark all the fields (or all highlighted fields) in the Field List.

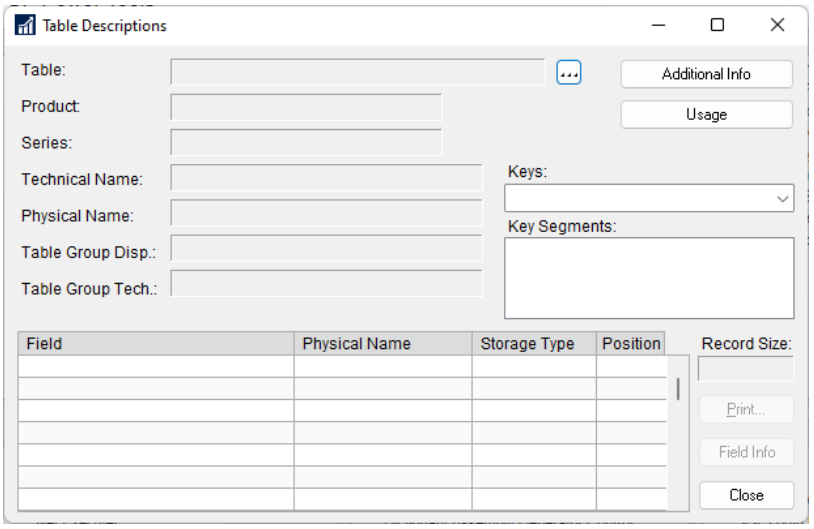
*Unmark All*

Use this button to unmark all the fields (or all highlighted fields) in the Field List.

The following is a description of the Options menu available for the window:

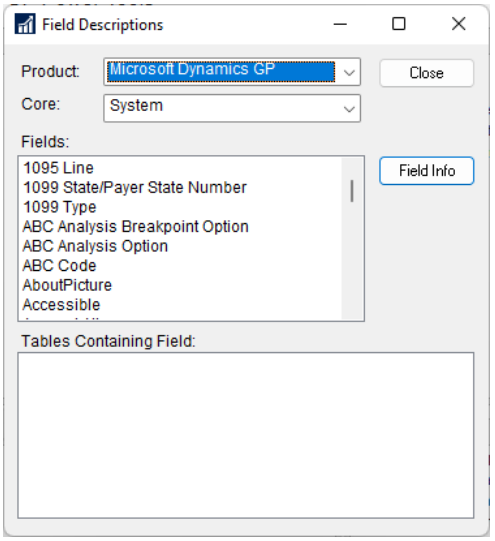
### *Table Descriptions*

Use this menu option to open the default Table Descriptions window. This also allows this window to be opened on the Web Client.



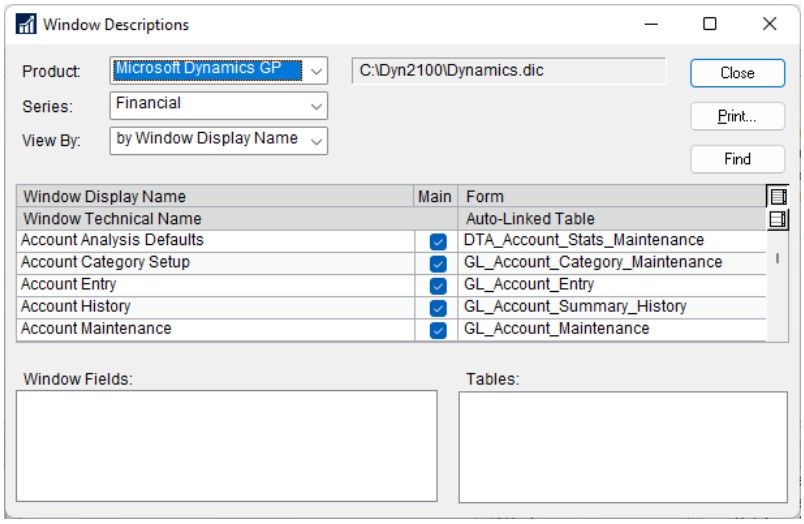

### *Field Descriptions*

Use this menu option to open the default Field Descriptions window. This also allows this window to be opened on the Web Client.



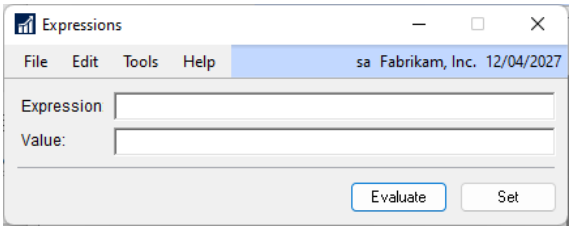
### *Window Descriptions*

Use this menu option to open the default Window Descriptions window. This also allows this window to be opened on the Web Client.



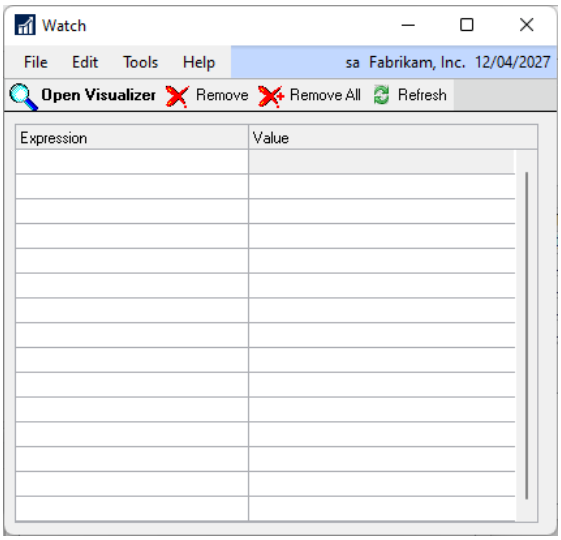

### *Debug Expressions*

Use this menu option to open the Script Debugger Expressions window. When the Expressions window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



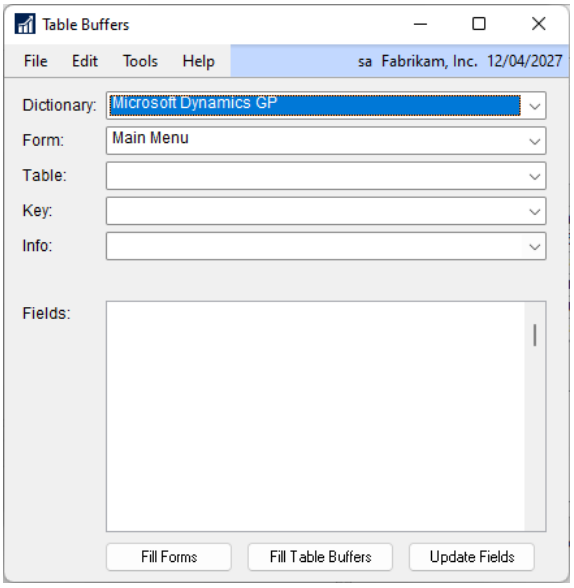
### *Debug Watch*

Use this menu option to open the Script Debugger Watch window. When the Watch window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.




### *Debug Table Buffers*

Use this menu option to open the Script Debugger Table Buffers window.

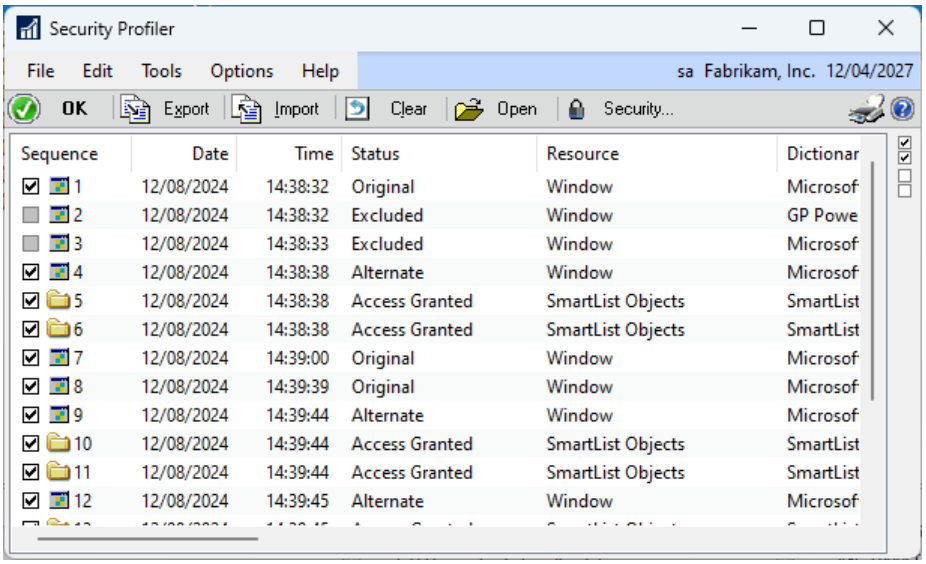



*Using the Find by Field Data mode without any Field or Table filters will scan all fields and tables in a database for the data being searched for. This query can take a long time to run and place a performance load on the SQL Server which could affect other users or processes. To minimize the impact on the system this mode can only be used by Administrator users.*

## Security Profiler

You can open the Security Profiler window by selecting Security Profiler from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Profiler from the Options button drop list on the main window.

After it has been opened, the Security Profiler window will monitor all application-level security requests and display the results.



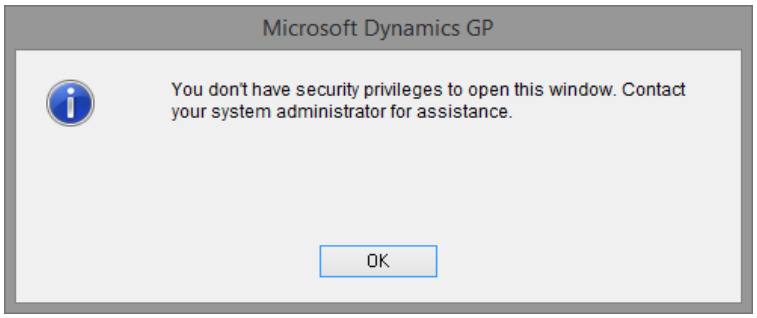

Whenever a form or report is opened, the application-level security is checked to confirm that the current user has access. Security is also checked to find out whether a customized version (modified, alternate or modified alternate) of the form or report is to be used.

When a report is opened, access is checked for all of the tables linked to the report. To be able to print the report, access must be permitted for the report itself and all the tables linked to the report.

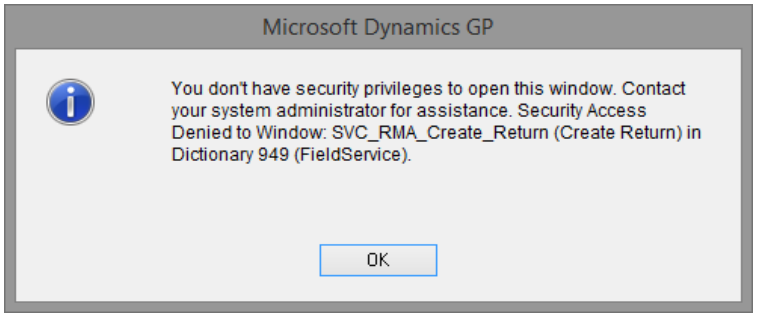
The Security Profiler will also track access to non-dictionary resource Security Objects, such as Customization Tools, Document Access, Letters, Microsoft Dynamics GP Import, Navigation Lists, Series Posting Permissions, and SmartList Objects. If the products are installed, the following objects are also supported, SmartList Builder Permissions and Extender Resources. Security objects from other 3rd party products will show as Unknown Objects.

The Security Profiler window displays each of the queries to the application-level security system and displays the results with all the relevant details of the resources involved.

The Security Profiler window can be used to identify which form or report is causing unexpected security privileges or access denied errors. Just open the Security Profiler and then perform the action in Microsoft Dynamics GP that causes the error to appear. The details of the resource causing the error will be displayed.




By default, GP Power Tools will append additional details to the dialog to identify the resource. This functionality can be disabled from the Administrator Settings window, if desired.




*The Security Profiler window only monitors application-level security. It will not display security issues caused by Windows security or SQL Server security.*

Below is a description of the individual fields on the window:

### *OK Button*

This button will close the Security Profiler window.

### *Export Button*

This button will allow a Security Profiler log to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window. This allows a user to provide all of the details of a security issue to the administrator for their analysis.

### *Import Button*

This button can be used to import a previously exported Security Profiler log. This allows an administrator to view a log of security issues provided by a user.

### *Clear Button*

This button can be used to clear the current contents of the Security Profiler window.

### *Open Button*

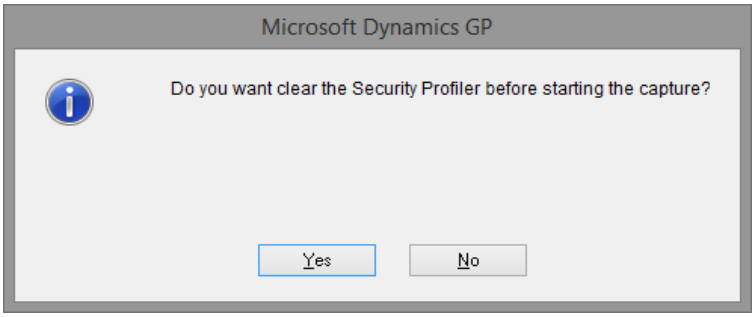
This button will open the selected form or report resource.

*Reports opened in this way will not have any options or restrictions applied and might contain unpredictable results. If the report uses a temporary table, this table will contain no data. Opening forms and reports from this window is only for testing purposes.*

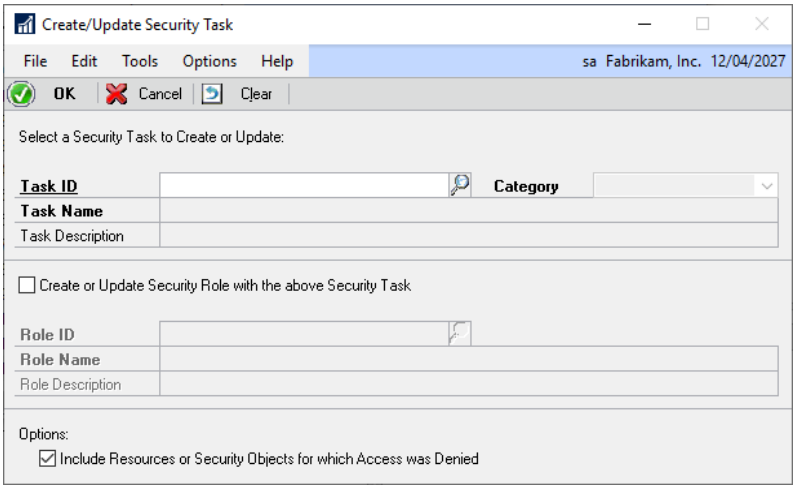
### *Security Button Drop List*

This button Drop List has the option to open the Security Information window for the selected resource. See sections below for more information. The Security Button will only be available if the current user has security access to the Security Information window.

If the current user has access to the Security Task Setup window, the option to Start Capture of Resources and Security Objects will be available. This option will offer to clear the Security Profiler if it is not empty:




If the current user has started the capture of Resources and Security Objects, the option to Stop Capture and create/update Security Task will be available. When this option is selected it will open the Create/Update Security Task window.



This window can be used to create a new Security Task or update an existing Security Task with the items listed in the Security Profiler. If the user has access to the Security Role Setup window, the option to create a new Security Role or update an existing Security Role with the Security Task ID will be available.

*Use the options to capture Resources and Security Objects and then create or update a Security Task based on the captured items to quickly build Security Tasks for specific activities within Microsoft Dynamics GP.*

### *Print Button*

This button will allow a report of the contents of the Security Profiler window to be printed.

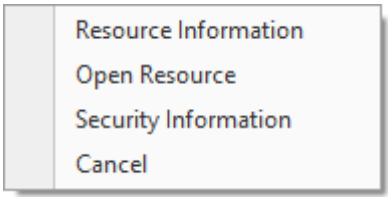
### *Mark All Button*

This button will select all the resources (or highlighted resources) on the window.

### *Unmark All Button*

This button will unselect all the resources (or highlighted resources) on the window.

*The Security Profiler window is Right click enabled. If you right mouse click on an item in the list you can select Resource Info (same as double click), Open Resource (same as Open Button), Security Info (same as Security Button) or Cancel from the context sensitive menu. The Security Info option will only be available if the current user has security access to the security windows under Tools >> Setup >> System.*




*The Security Profiler window can be configured to open automatically when there is a security issue. This option is controlled from the Administrator Settings window.*

*The Security Profiler window has an Options Menu which can be used to Refresh Application Navigation. This option can be used by a user to update the application's navigation menus to reflect changes made to security without having to exit and re-launch the application.*

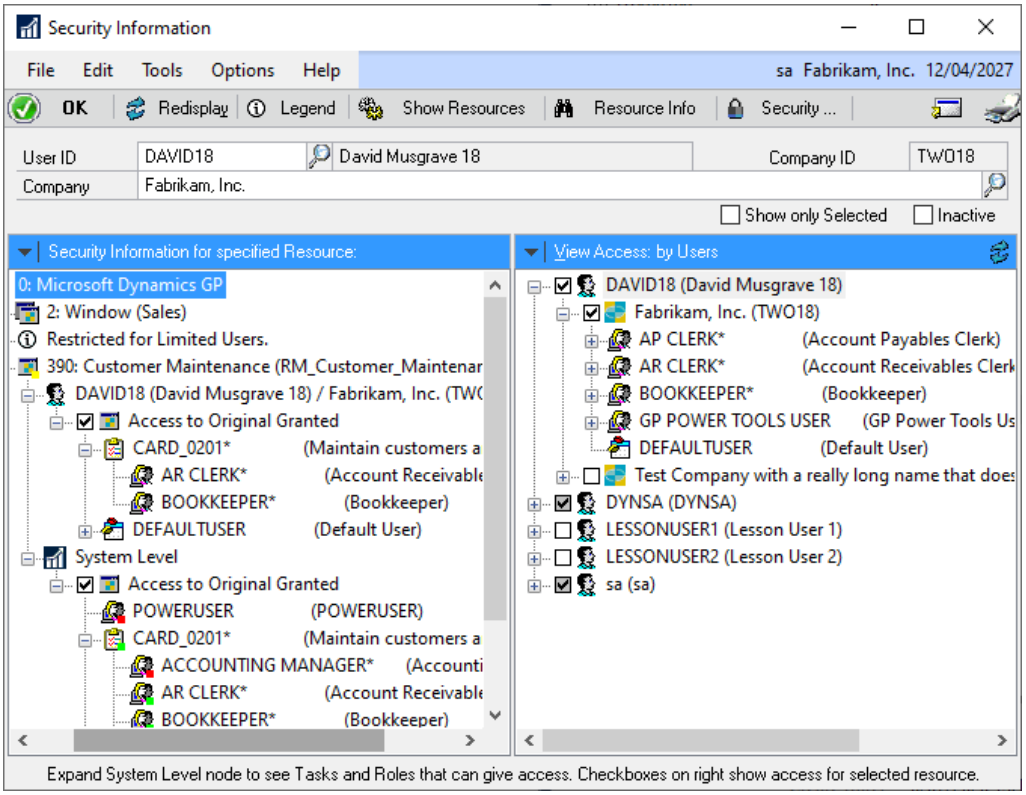
## Security Information

You can open the Security Information window by selecting Security Information from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Information from the Options button drop list on the main window. Once opened, you can use the drop-down menu on top of the left pane to select a resource. You may select a Form (by Dictionary or by Menu) as well as a Table or Report resource, a Security Object, or a Service Enabled Procedure.

You can also open the Security Information window from the Resource Information window or the Security Profiler window. From these windows use the Security Button or the Security Info option from the local context (right click) menu to show security information for the selected resource.

The Security Information window is designed to display the security settings for the selected resource for a particular user and company combination. Once the information is displayed the administrator can use the Go To Button or double click to open the appropriate security administration window to make changes if necessary.

Below is an example of the Security Information window. It shows the security settings for the user including the security tasks that belong to security roles assigned to that user. Also shown is the alternate/modified form and report ID to show which version of a resource the user has access to. Under the System Level node, all security tasks, security roles and alternate/modified form and report IDs which reference the selected resource are displayed.



The tree in the left-hand pane is used to display the security status for the currently selected user and company for the selected security resource. The first 3 nodes of the tree describe the product dictionary, resource type (and series) and resource by Display and Technical Name.

*If a resource is not available on the Web Client or not available to Limited Users, this will be displayed on an information node on the tree. Also a Limited User will be highlighted with a yellow dot on the icon and Limited User in the description.*

The next section is the User node which shows if the user has access to the current resource and which Security Tasks and Security Roles provided that access. If the resource is a Form or Report, the Alternate Modified Form and Report ID will be shown to define which version of the resources access is granted to.

The third section is the System node which shows all Security Tasks and Security Roles which reference the current resource and all Alternate Modified Forms and Report IDs that reference the current resource. Security Tasks, Security Roles and Alternate Modified Forms and Reports IDs in this view will have a green or red indicator to show whether the current user and company has access.

*Expand the System Level node on the left-hand pane to see what tasks are assigned to the currently selected resource or operation. Expand the tasks to see what roles can be used to give a user access to the currently selected resource or operation. If the only task and role available is POWERUSER, then the current resource or operation has not been added to any tasks.*

Below is a description of the individual fields on the window:

*User ID*
This is the User ID for which security is being checked.

*Company*
This is the company for which security is being checked.

*Show only Selected*
When this checkbox is selected, only users with access will be shown.

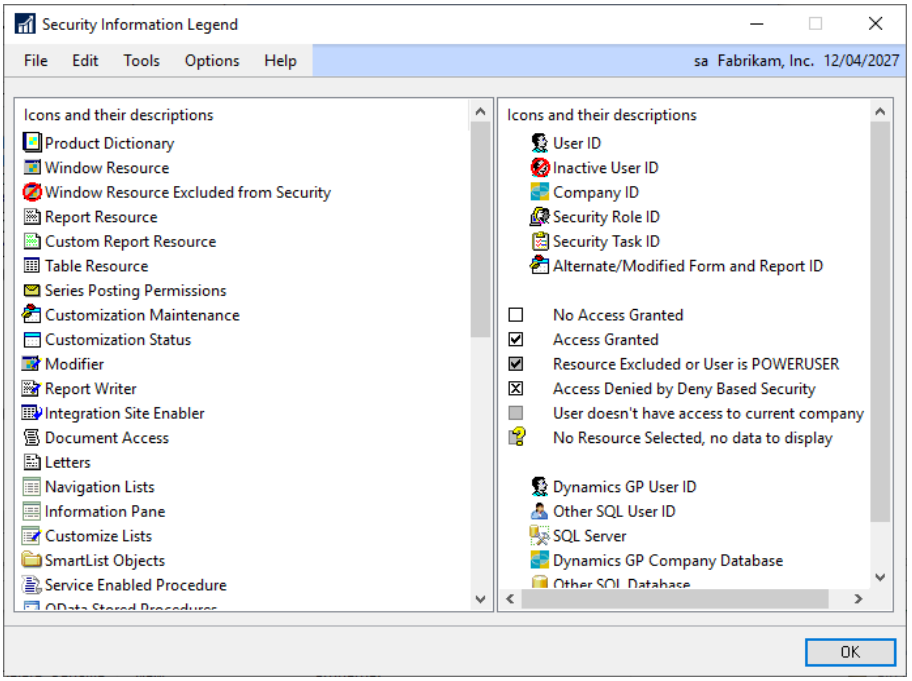
*Inactive*
When this checkbox is selected, inactive users will be shown.

*OK Button*
This button will close the Security Information window.

*Redisplay Button*
This button will re-populate the security information tree. Use this button after making security changes to see the new updated security.

### *Legend Button*

This button will open the Security Information Legend window to show the meanings of the different icons used.




### *Show Resources Button*

This button will open the Security Information Resources window.

### *Resource Info Button*

This button will open the Resource Information window.

### *Security Button*

Use this button to access Deny Based Security and select from the Enhanced Security window, the Security Denied window and the Security Hidden window.

### *Go To Button*

This button allows the user to open a system security window.

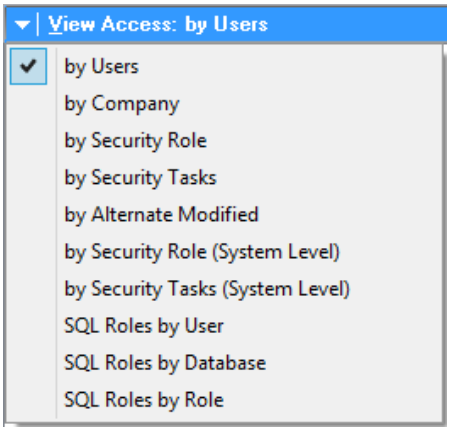
### *Print Button*

Use this button to print a report of the user and company access for the currently selected resource with details of which Security Roles and Security Tasks granted access.

*You can double click on the User ID/Company node to open the User Security Setup window; a Security Task ID to open the Security Task Setup window; a Security Role ID to open the Security Role Setup window; and an Alternate/Modified Forms and Reports ID to open the Alternate/Modified Forms and Reports window.*

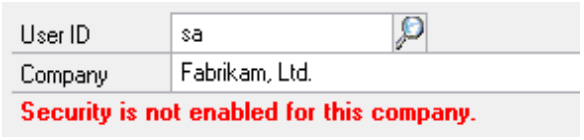
The right-hand pane on the Security Information window displays several different views into the company access and security information. Use the View Access button drop-down list to change view. When changing views, the currently selected object will remain selected if possible. This pane can be used even when no resource is selected before opening the Security Information window.

Below are the views available.




These views will provide a visual representation of the relationships between Security Tasks, Security Roles, Alternate Modified Forms and Report IDs, Users and Companies.

*The Security Information window will highlight when security is not activated for the selected company. This can be enabled from Company Setup window (Microsoft Dynamics GP >> Tools >> Setup >> Company >> Company).*



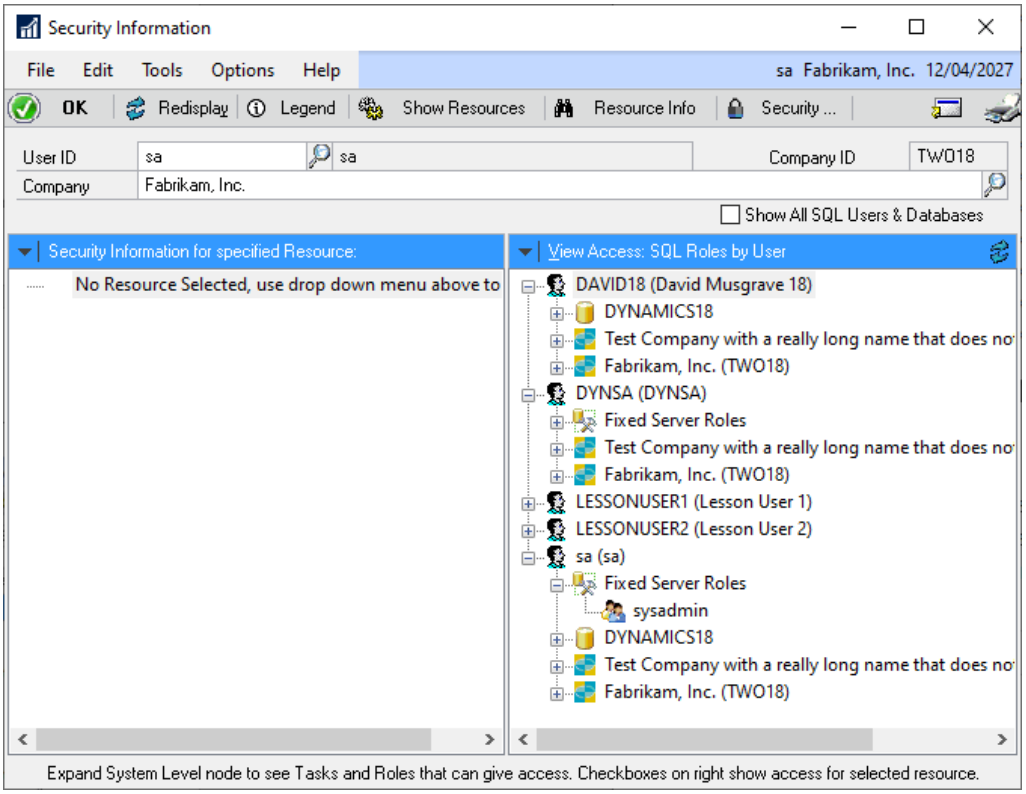

*The Security Information window is Splitter enabled which allows the ratio between the left and right-hand panes to be adjusted. When running on the Web Client, the splitter functionality is disabled..*

## Security Information SQL Role Views

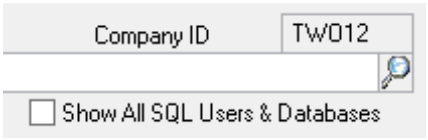
The Security Information window can also be used to show the SQL Server Roles assigned to users at the SQL Server level as well as for each database. There are three views available to view the data by Users, by Database and by Role.

*The SQL Role information is read from the SQL Server the first time one of the three SQL Role views is selected. On a large system, there might be a small delay while the data is read from the SQL Server. To force the data to be read again, close and re-open the Security Information window.*

Below is an example screenshot.




The data shown in the three views is restricted to only include Dynamics GP users and databases by default. To show all users and database, select the Show All SQL Users & Databases checkbox.




Once the option has been selected, the view will be refreshed to include the additional data for non-Dynamics GP users and databases.

## Security Information Resources

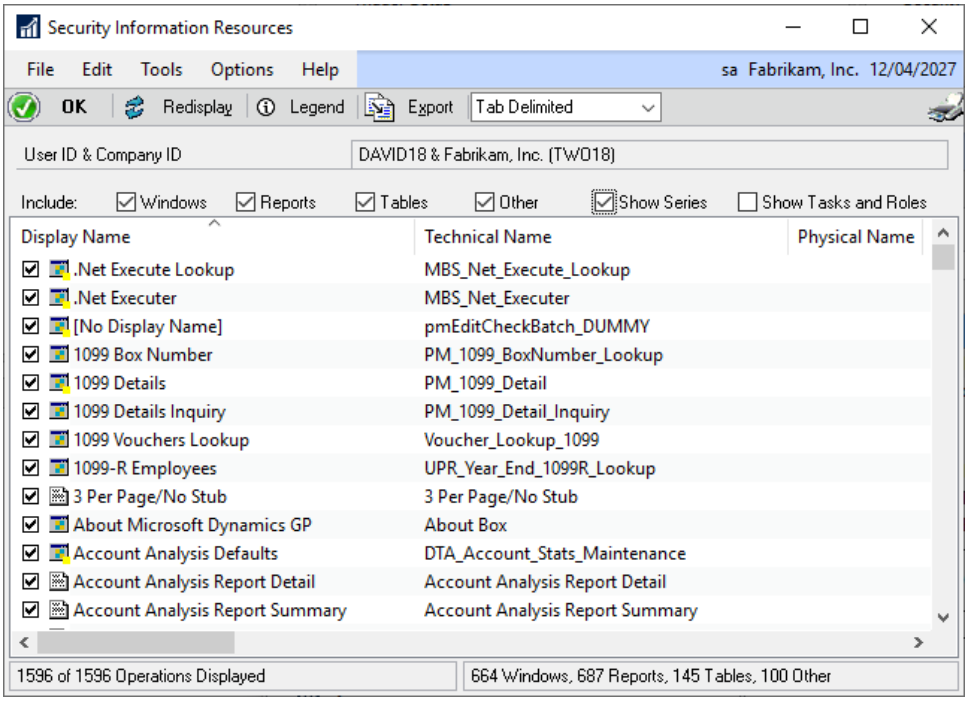
When the Show Resources Button is clicked, the Security Information Resources window will open.

This window will display the resources associated with the currently selected User ID/Company combination, Security Role ID, Security Task ID or Alternate/Modified Forms and Report ID in the right-hand pane of the Security Information window. Changing the selection will cause the window to refresh.

You can use the check boxes to decide which resource types (Forms, Reports, Tables and Other) to include in the displayed resources. These selections can be changed while the window is populating.

The resources displayed are those for which the selected User ID/Company combination, Security Role ID, Security Task ID or Alternate/Modified Forms and Report ID has access to.

If the selected node in the right-hand pane of the Security Information window has a User ID and/or Company ID parent node, the system will be able to identify which Alternate/Modified Forms and Report ID to apply and so will display when an alternate and/or modified version of the resources has been selected.




Below is a description of the individual fields on the window:

*OK Button*

This button will close the Security Information Resources window.

*Redisplay Button*

This button will re-populate the window. Use this button after making security changes to see the new updated security.

*Legend Button*

This button will open the Security Information Legend window.

*Export Button*

This button will allow the resources displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

*Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

*Show Series*

Use this checkbox if you want the series information included in the resource list.

*Display Security Tasks and Roles*

Use this checkbox if you want the Security Tasks and Security Roles displayed in the resource list. When this option is selected, multiple lines will be displayed for resources if there are more than one Security Task or Security Role which provides access to the resource.

*Print Button*

A report of the contents of the resource list can be printed using this button.

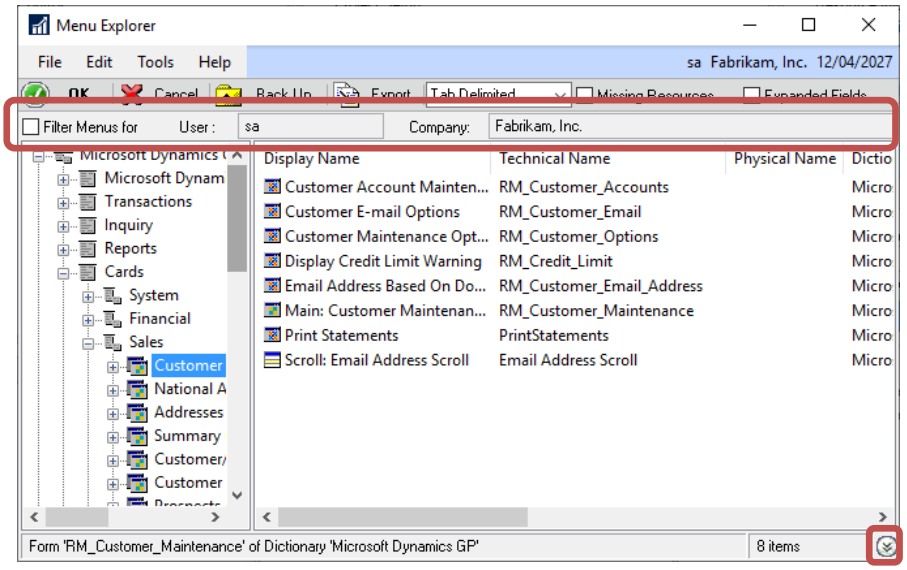
*When opening the Security Information window, a background process is launched to check if all the dictionary resources and security objects have been added to the syCurrentResources (SY09400) table. If information is found to be missing or dictionaries have been added or updated, GP Power Tools will generate the additional data. GP Power Tools will also add the additional data when the table is cleared using the Clear Data window.*

*Once all the dictionary resources and security objects have been added to the syCurrentResources (SY09400) table, GP Power Tools will create a SUPERUSER Security Task with access to everything and a matching SUPERUSER Security Role. Using the SUPERUSER Security Role is similar to the POWERUSER Security Role but uses the security model rather than bypassing it. GP Power Tools will keep the SUPERUSER Security Task updated automatically.*

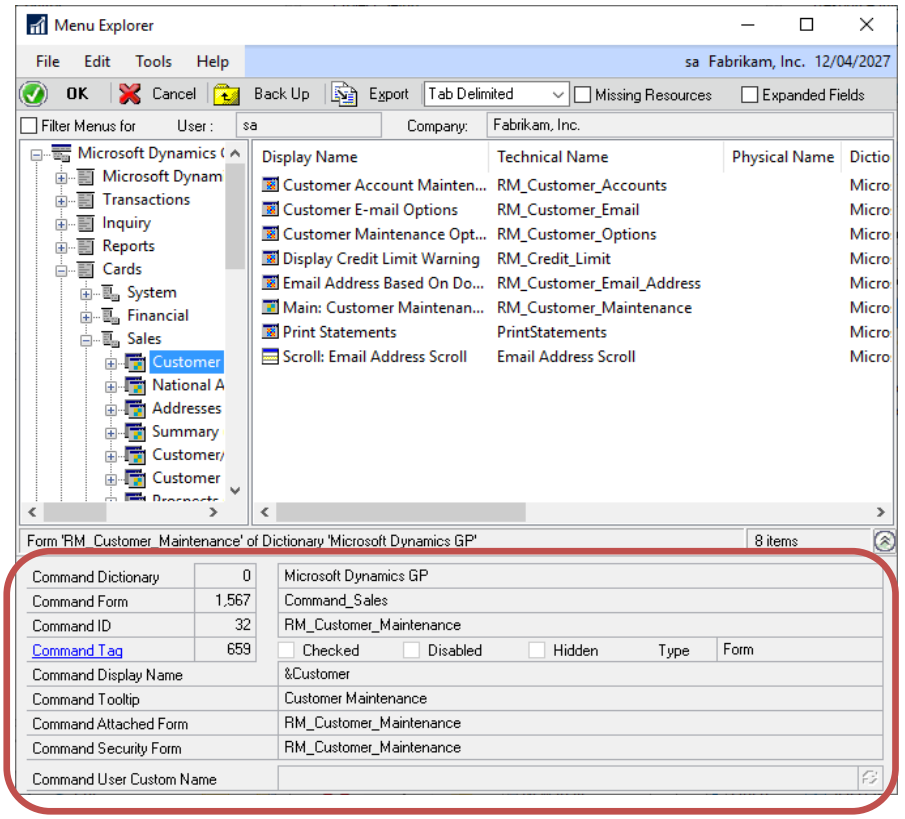
*The Security Information window has an Options Menu which can be used to Refresh Resource Information Table. This option can be used by a user to clear and then update the syCurrentResources (SY09400) table without having to use the Clear Data window. The SUPERUSER Security Task and SUPERUSER Security Role will also be updated using this option.*

The following section covers additional functionality available in the Menu Explorer window.

When the Menu Explorer opened from the Security Information window, you have the option to filter the menus for the current user and company based on their security access.



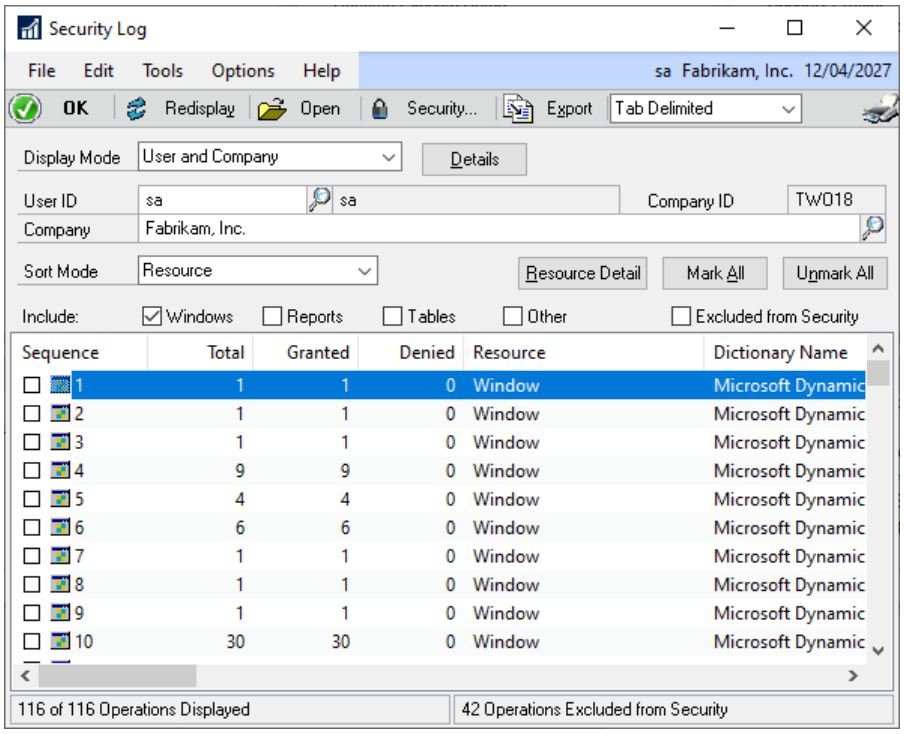

The Menu Explorer also has the option to display the details for the menu command, by click the expansion button (shown above). This will open the Menu Command Details expansion area at the bottom of the window.



## Security Log

You can open the Security Log window by selecting Security Log from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Log from the Options button drop list on the main window.

The Security Log window displays the data captured by the Security Activity Tracking option which can be enabled from the Administrator Settings window using the Enable Security Activity Tracking option.




Once the Security Activity Tracking is enabled, all security events (both granted and denied) are tracked. The logging does not track individual events, but instead totals up the number of events so you can see which resources are accessed the most. It also tracks the last three security events for a resource.

Each event is tracked for the user and company, user, company and system wide, and you select how you want to view the data.

You can use the check boxes to decide which resource types (Forms, Reports, Tables and Other) to include in the displayed resources. These selections can be changed while the window is populating.

Below is a description of the individual fields on the window:

*Display Mode*

This drop-down list allows you to select whether your wish to view data for the selected user and company, for a specific user or company or for all users and companies.

*User ID*

Use this field to select the User ID to display.

*Company*

Use this field to select the Company to display.

*Sort Mode*

This drop-down list can be used to select the order that the Security Log entries are initially displayed in. You can also adjust the sort after the data is displayed by clicking on the column headers.

*Excluded from Security*

This checkbox can be selected if you wish to see the resources which have been accessed which are excluded from the application security system.

*OK Button*

This button will close the Security Log window.

*Redisplay Button*

This button can be used to redisplay the current contents of the Security Log data to the window.

*Open Button*

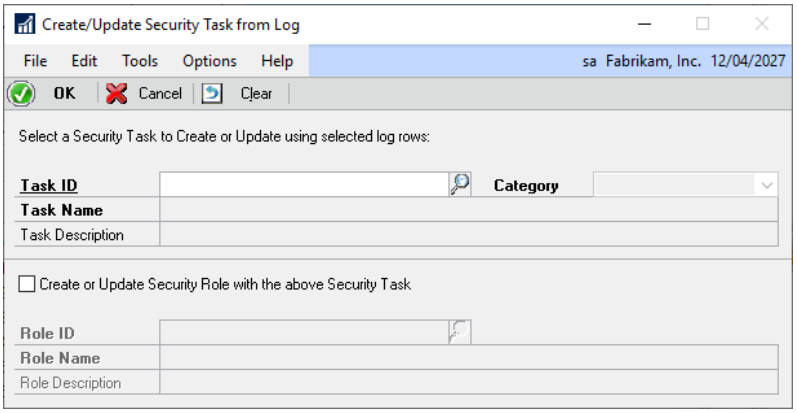
This button will open the selected form or report resource.

*Reports opened in this way will not have any options or restrictions applied and might contain unpredictable results. If the report uses a temporary table, this table will contain no data. Opening forms and reports from this window is only for testing purposes.*

*Security Button Drop List*

This button Drop List has the option to open the Security Information window for the selected resource. See sections below for more information. The Security Button will only be available if the current user has security access to the Security Information window.

The option to Create/update Security Task from selected rows will open the Create/Update Security Task from Log window.



This window can be used to create a new Security Task or update an existing Security Task with the selected rows listed in the Security Log window. If the user has access to the Security Role Setup window, the option to create a new Security Role or update an existing Security Role with the Security Task ID will be available.

*Use the options to capture Resources and Security Objects and then create or update a Security Task based on the captured items to quickly build Security Tasks for specific activities within Microsoft Dynamics GP.*

### *Export Button*

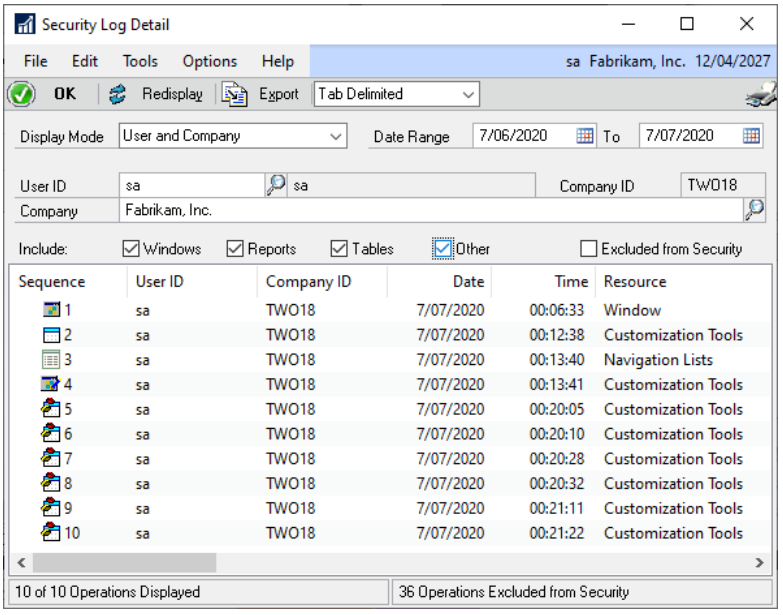
This button will allow the result set displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

### *Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

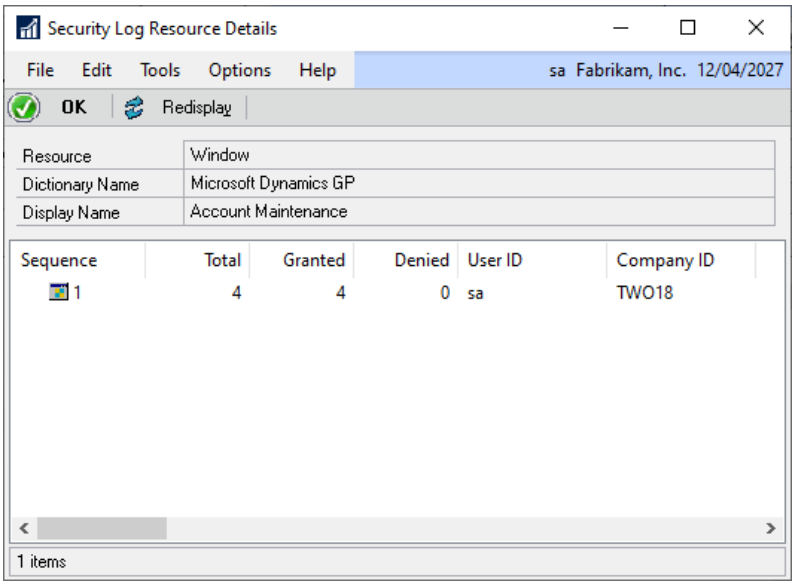
### *Details Button*

This button will open the Security Log Detail window to display individual records of each security event. Turn the capture of this detailed data on from the Administrator Settings window.



### *Resource Detail Button*

This button will open the Security Log Resource Details window to display other log entries for the currently selected resource. This allows you to easily check which other users and/or companies are using a resource.



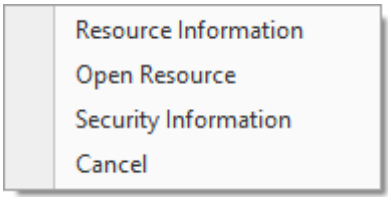

### *Mark All Button*

This button will mark all lines (or all highlighted lines) in the current Security Log view as selected.

### *Unmark All Button*

This button will mark all lines (or all highlighted lines) in the current Security Log view as unselected.

*The Security Log window is Right click enabled. If you right mouse click on an item in the list you can select Resource Info (same as double click), Open Resource (same as Open Button), Security Info (same as Security Button) or Cancel from the context sensitive menu. The Security Info option will only be available if the current user has security access to the security windows under Tools >> Setup >> System.*




*To clear the data in the Security Log table to start capturing data again, use the Configuration Maintenance window to clear the data in the MBS_SecurityLog table.*

## Security Analyzer

The Security Analyzer is a tool for administrators to analyze the security settings of their Microsoft Dynamics GP system. It is designed to highlight potential security risks, provide information on unused settings as well as provide a comparison between security access and security resources actually used.

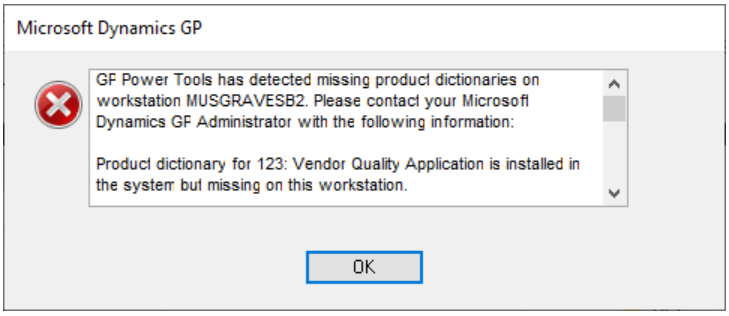
The Security Analyzer has over twenty queries which can be displayed in both Summary and Detail formats. The queries are divided into System Level queries and User & Companies queries. For the Users & Companies queries, you can select the User and/or Company to limit the query data to.

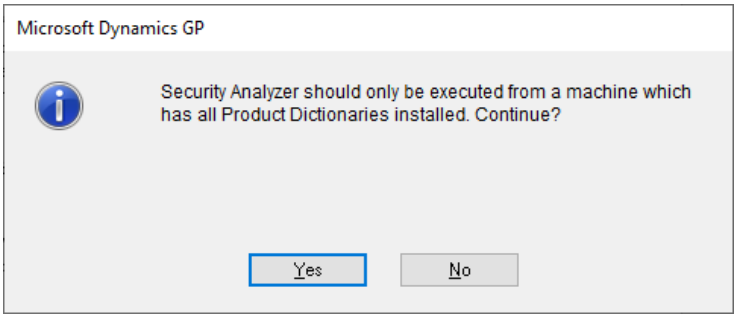
Some of the queries in the Security Analyzer window use the Security Log window's activity data captured by the Security Activity Tracking option which can be enabled from the Administrator Settings window using the Enable Security Activity Tracking option.

Once the data is displayed, it can be exported if desired, or used to drill down to the relevant system windows to be able to make changes to the system and security settings.

You can open the Security Analyzer window by selecting Security Analyzer from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Analyzer from the Options button drop list on the main window.

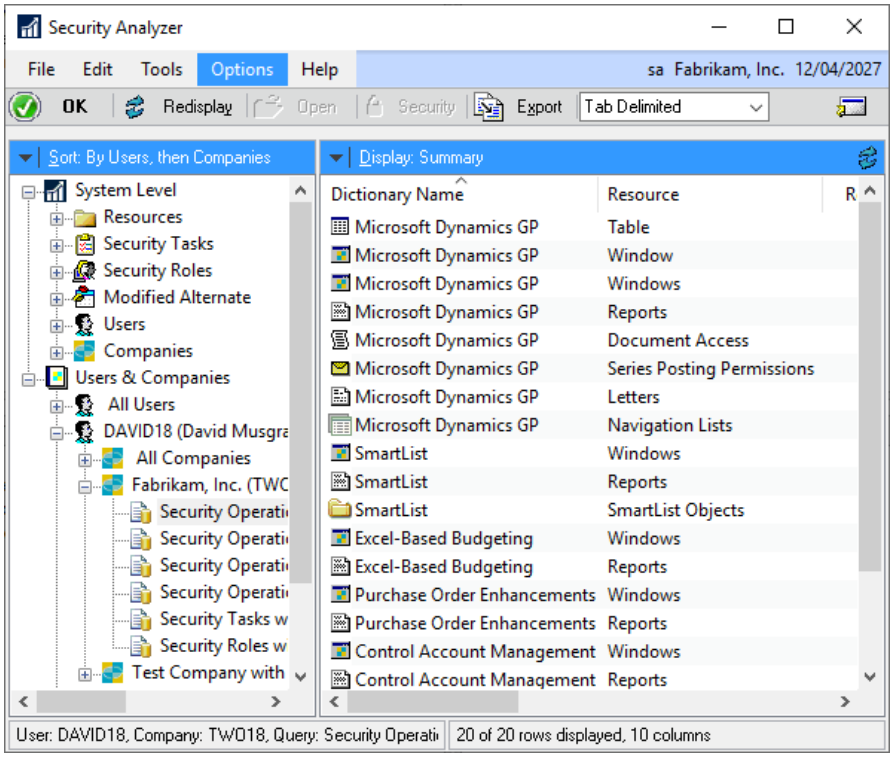
Before the window opens the system will check if you have the dictionaries for all products installed. If not, the following dialogs will be displayed with the details of the missing products. This is to ensure that the security data in the system will be valid for all installed products.



*When opening the Security Analyzer window, it will check if all the dictionary resources and security objects have been added to the syCurrentResources (SY09400) table. If information is found to be missing or dictionaries have been added or updated, GP Power Tools will generate the additional data. GP Power Tools will also add the additional data when the table is cleared using the Clear Data window.*

After confirming all the products are installed and the updating the security resources table has completed, the Security Analyzer window will open.




Below is a description of the individual fields on the window:

*OK Button*

This button will close the Security Analyzer window.

*Redisplay Button*

This button will re-populate the window. Use this button after making changes to users and companies. To refresh the current query, use the Refresh button in the top right corner of the right-hand pane.

*Open Button*

This button will open the selected form or report resource.

*Security Button*

This button will open the Security Information window for the selected resource.

*Export Button*

This button will allow the resources displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

*Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

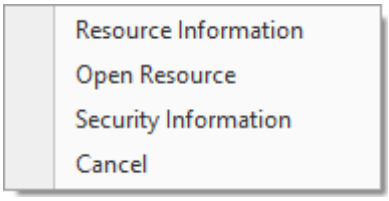
*Go To Button*

This button allows the user to open a system security window.

Select the query you wish to view using the left-hand tree pane and it will be displayed in the right-hand list pane. You can change the order that Users and Companies are displayed in using the view button above the left-hand pane. You can also swap between Summary and Detail view using the view button above the right-hand pane. Double clicking on the data in a Summary view in the right pane will jump to the Detail view of the same query.

*You can double click on the User ID/Company node to open the User Security Setup window; a Security Task ID to open the Security Task Setup window; a Security Role ID to open the Security Role Setup window; and an Alternate/Modified Forms and Reports ID to open the Alternate/Modified Forms and Reports window.*

*The Security Analyzer window is Right click enabled. If you right mouse click on an item in the list you can select Resource Info (same as double click), Open Resource (same as Open Button), Security Info (same as Security Button) or Cancel from the context sensitive menu. The Security Info option will only be available if the current user has security access to the security windows under Tools >> Setup >> System.*




*The Security Analuyzer window is Splitter enabled which allows the ratio between the left and right-hand panes to be adjusted. When running on the Web Client, the splitter functionality is disabled..*

*The Security Analyzer window has an Options Menu which can be used to Refresh Resource Information Table. This option can be used by a user to clear and then update the syCurrentResources (SY09400) table without having to use the Clear Data window.*

## Deny Based Security – Introduction

The security system in Microsoft Dynamics GP controls access to all resources within the application. This includes dictionary resources such as Forms, Reports and Tables as well as other security objects such as Document Access and Posting Permissions.

For Microsoft Dynamics GP prior to Version 10.0, the security model was an optimistic user & company and class-based design. This design meant that a user had access to every resource within the application unless it was specifically denied. The settings for a user could be set at the user & company level or set a class level and rolled down to users belonging to that class.

From Microsoft Dynamics GP Version 10.0, the security model was changed to a pessimistic task and role-based design. This design meant that a user had no access to any resources unless they were specifically granted to them. Granting access worked by grouping multiple resources or operations needed to perform a function into tasks. Multiple tasks could then be assigned to roles. Finally, a user could be assigned to multiple roles depending on the work they do within each company.

The task and role-based model works really well to grant access to users based on the work they do but does not easily allow for minor differences between users.

*For example: If you have two users with the same roles but wish to deny access to a single window from one user, the process to remove access to one resource is cumbersome and difficult to maintain. You would need to duplicate any task that gave access to the window and remove that one window from the now duplicated tasks, then you would need to duplicate any role that linked to any of the original tasks now updated and change them to use the duplicated tasks. Finally, you would need to assign the now duplicated roles to the user.*

Based on the above example, you can see that over time, your security data would be filled with duplicated tasks and roles without only minor differences between them and no easy method to compare the differences.

There must be a better way.... Introducing Deny Based Security.

Deny Based Security adds an optional additional layer to the Microsoft Dynamics GP security model, which allows individual resources or operations to be denied on a per user & company basis regardless of what has been granted by the task and role model. Once a resource is marked as denied for a user & company combination, access will never be available for that user & company.

Deny Based Security also adds the ability to hide items from the menu navigation when those items cannot be controlled by security. This works for both menu items linked to forms excluded from security and menu items which run scripts rather than opening forms.

*The Security Denied functionality, once applied, works whether GP Power Tools is installed or not. The Security Hidden functionality does require the GP Power Tools to remain installed (which is the recommended configuration anyway).*

## Deny Based Security – Enhanced Security

You can open the Enhanced Security window by selecting Enhanced Security from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Enhanced Security from the Options button drop list on the main window.

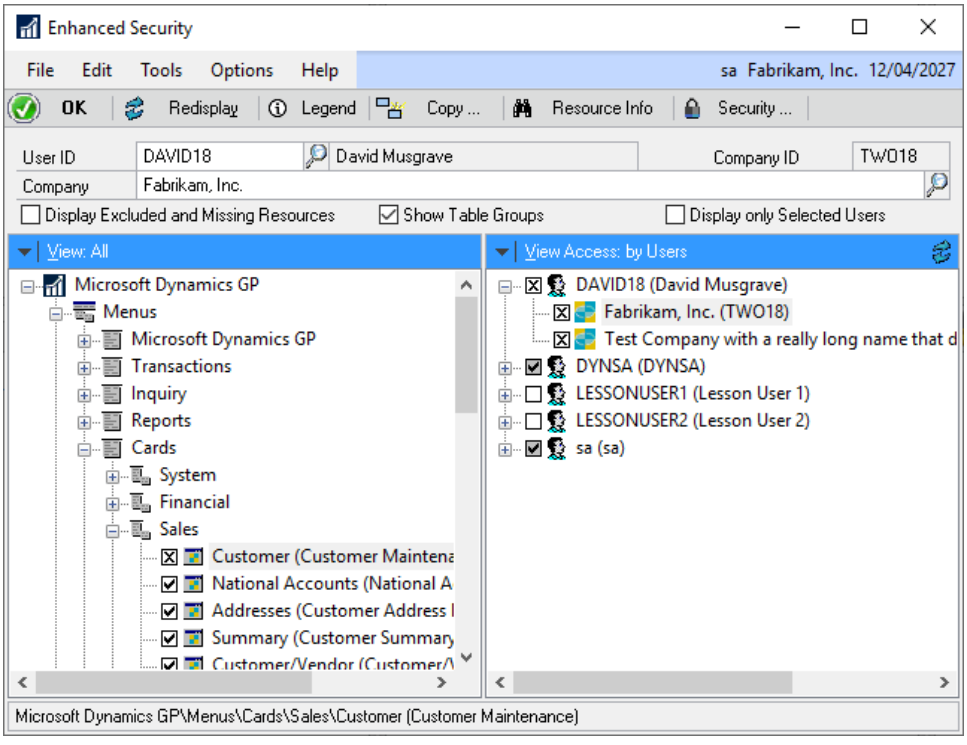
The Enhanced Security window allows you view the security resources or operations via the navigation model (Menus or Area Pages) or via the dictionary model. You can also view security resources contained in specific Security Roles or Security Tasks.

The status icons on the left-hand pane show the security status for the user & company selected in the right-hand pane. The status icons on the right-hand pane show the security status for the resource or operation selected in the left-hand pane.

*Checkboxes are used for security resources or operations which can be controlled by the Security Denied functionality. Radio Buttons are used for menu commands which can be controlled by the Security Hidden functionality. A checkbox or radio button will show as disabled if the resource is excluded from security or the user belongs to the POWERUSER role.*

Clicking on an enabled status icon will toggle the item as Security Denied or Security Hidden accordingly. Changes are made immediately and do not need the OK Button to be clicked.

*Right clicking on the left-hand or right-hand panes while items are selected in the trees will open a context menu with the options to roll down changes to multiple users and/or companies.*



Below is a description of the individual fields on the window:

*OK Button*

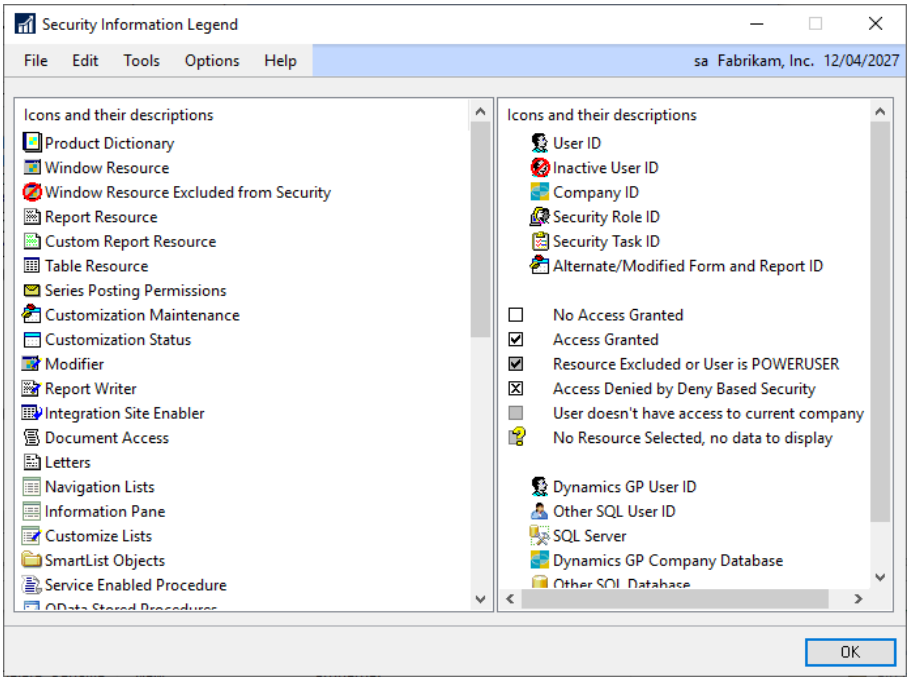
This button will close the Enhanced Security window.

*Redisplay Button*

This button will re-populate the left-hand pane of the window. To refresh the Users and Companies, use the Refresh button in the top right corner of the right-hand pane.

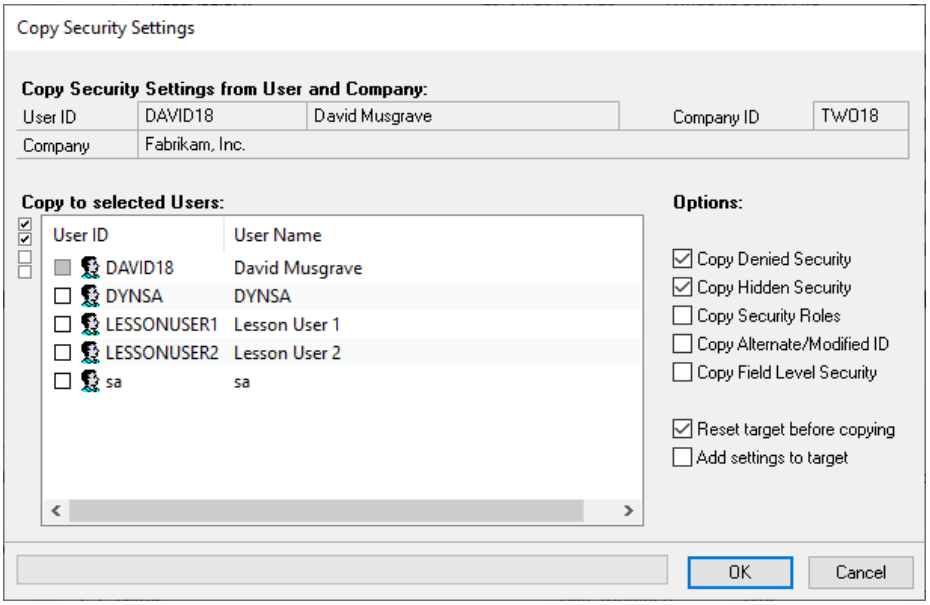
*Legend Button*

This button will open the Enhanced Security Legend window to show the meanings of the different icons used.

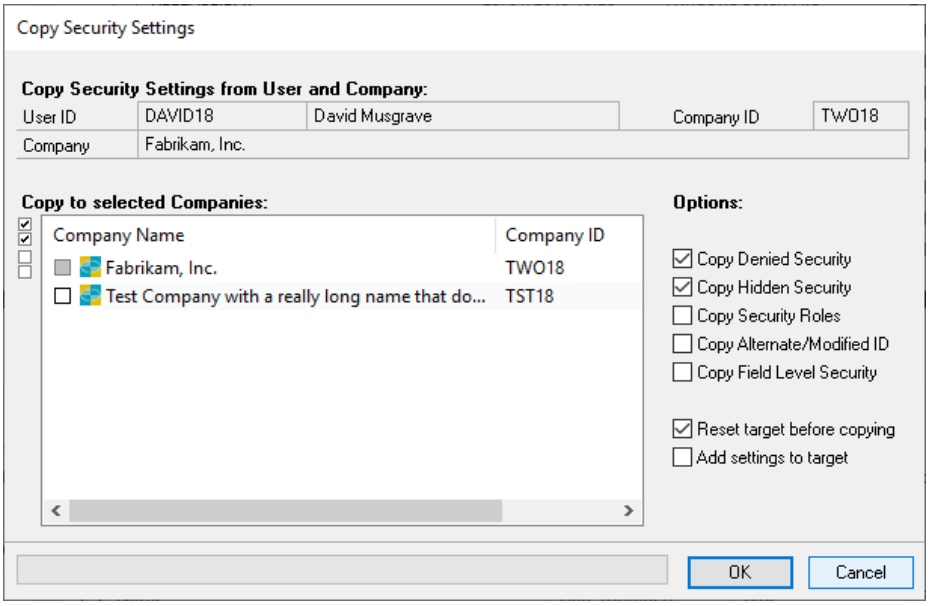



*Copy Button*

Use this button to copy security settings from the current to user & company to other users in the current company:




Or copy security settings from the current to user & company the current user in other companies:




*Use these windows to copy Security Denied and Security Hidden data to other users and companies in the system. For your convenience, these windows can also copy Security Roles and Alternate Modified Forms and Report IDs as well as Field Level Security settings. You have the option to make an exact copy (Reset target before copying) or combine the settings (Add settings to target).*

### *Resource Info Button*

This button will open the Resource Information window.

*Security Button*

Use this button to access the Security Information window or the other Deny Based Security windows and select from the Security Denied window and the Security Hidden window.

The following is a description of the fields on the window:

*User ID*

This is the User ID for which security is being displayed.

*Company*

This is the company for which security is being displayed.

*Display Excluded and Missing Resources*

Selecting this checkbox will show all resources in the left-hand pane, even if the resource is excluded from security or is missing.

*Show Table Groups*

Unselecting this checkbox will show the table resources under Dictionary Tables without using the table group logical tables.

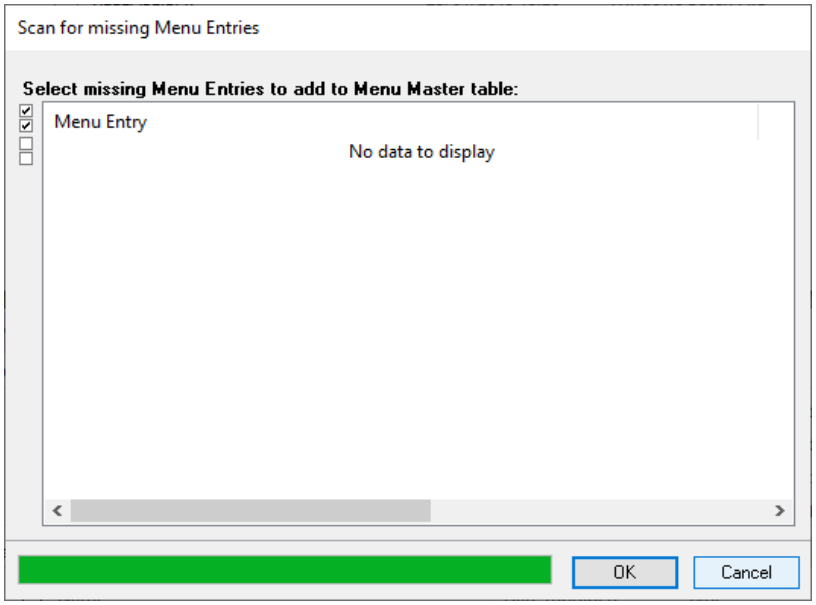
*Display only Selected Users*

Selecting this checkbox will only show users in the right-hand pane if they have access to the resource selected in the left-hand pane.

*The Enhanced Security window has an Options Menu which can be used to Refresh Application Navigation. This option can be used to update the application's navigation menus to reflect changes made to security without having to exit and re-launch the application.*

*The Menus navigation view uses the data in the Menu Master syMenuMstr (SY07110) table to create the view. Some products do not correctly add their menu items to the table and so their menus will not show on the view.*

*The Options Menu on Enhanced Security window can be used to Scan for missing Menu Entries. This option will open the following window can scan the menu navigation in memory and look for entries which exist in memory but not in the table.*




*You can then select the entries you would like to add to the table and these will show on the Enhanced Security Menus view next time it is used.*

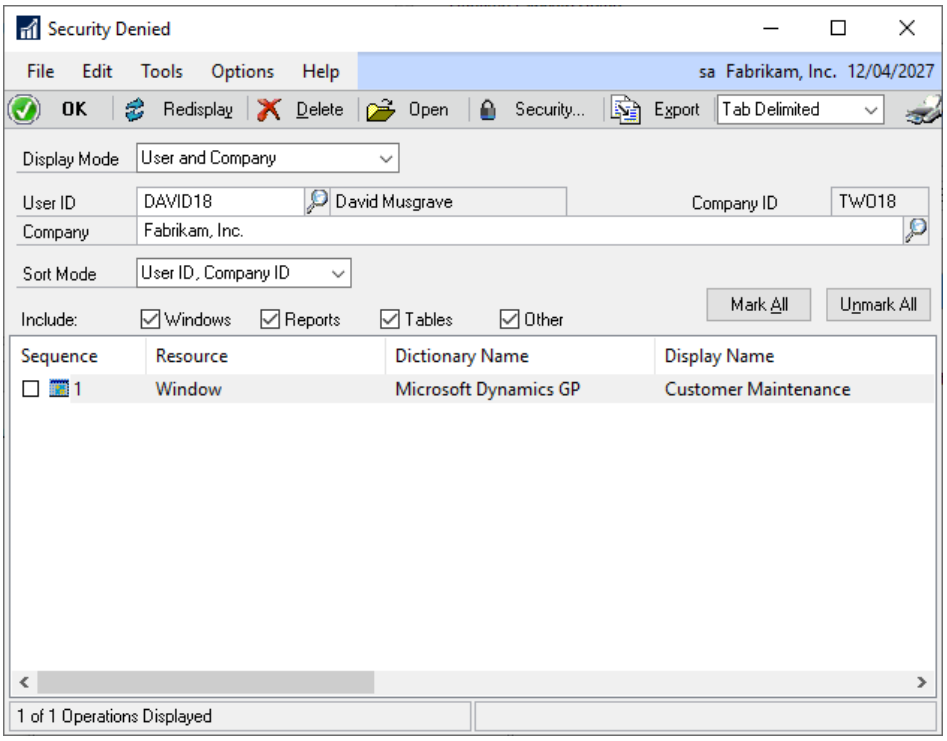
*When opening the Enhanced Security window, it will check if all the dictionary resources and security objects have been added to the syCurrentResources (SY09400) table. If information is found to be missing or dictionaries have been added or updated, GP Power Tools will generate the additional data. GP Power Tools will also add the additional data when the table is cleared using the Clear Data window.*

## Deny Based Security – Security Denied

You can open the Security Denied window by selecting Security Denied from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Denied from the Options button drop list on the main window.

The Security Denied window is used for maintenance, exporting and reporting of the Deny Based Security – Security Denied data. Using this window allows all the Security Denied for users to be easily viewed without having to explore the tree views on the Enhanced Security window.

Changes in the window are made immediately and do not need the OK Button to be clicked.




Below is a description of the individual fields on the window:

*OK Button*
This button will close the Security Denied window.

*Redisplay Button*
This button will re-populate the window.

*Delete Button*
This button will permanently remove the Security Denied records with a marked checkbox.

*Open Button*
Use this button to open the Window or Report currently selected.

*Security Button*
Use this button to access the Security Information window or the Enhanced Security window.

*Export Button*
This button will allow the resources displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

*Export Mode*
Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

*Print Button*
This button will allow a report of the contents of the Security Denied window to be printed.

The following is a description of the fields on the window:

*Display Mode*
Use this drop-down list in conjunction with the User ID and Company ID fields to control which records are displayed in the Security Denied view.

*User ID*
This is the User ID for which security is being displayed.

*Company*
This is the company for which security is being displayed.

*Sort Mode*
Use this drop-down list to control the order that the records are displayed in the Security Denied view.

*Include*
Use these resource type checkboxes to filter the records are displayed in the Security Denied view.

*Mark All Button*
This button will mark all lines (or all highlighted lines) in the current Security Denied view as selected.

*Unmark All Button*
This button will mark all lines (or all highlighted lines) in the current Security Denied view as unselected.

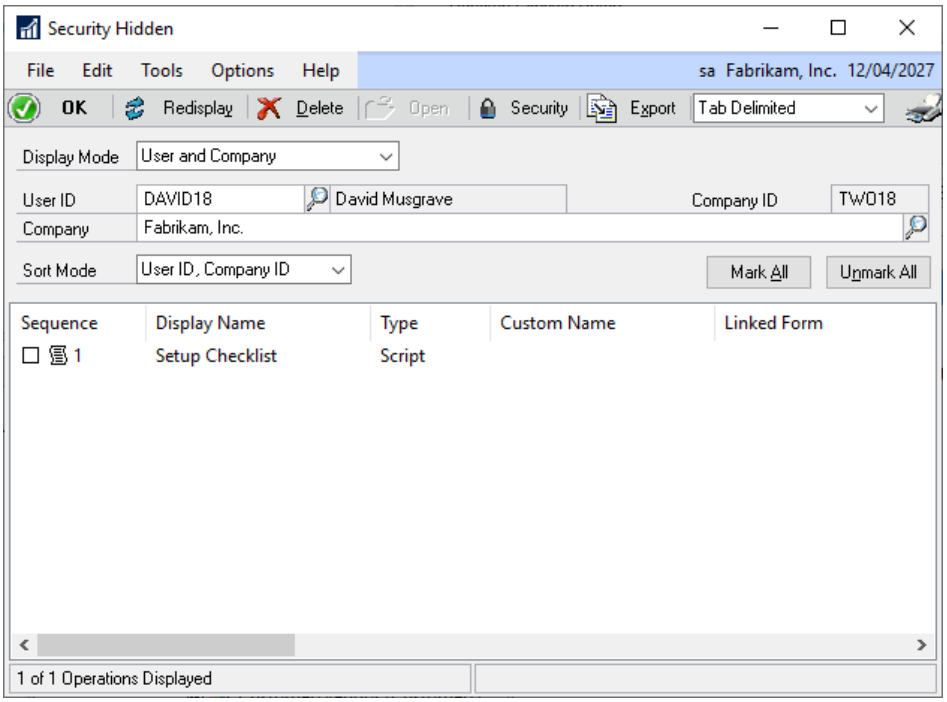
*The Security Denied window has an Options Menu which can be used to Refresh Application Navigation. This option can be used to update the application's navigation menus to reflect changes made to security without having to exit and re-launch the application.*

## Deny Based Security – Security Hidden

You can open the Security Hidden window by selecting Security Hidden from the Reports section of the GP Power Tools Area Page or by selecting Resources and Security >> Security Hidden from the Options button drop list on the main window.

The Security Hidden window is used for maintenance, exporting and reporting of the Deny Based Security – Security Hidden data. Using this window allows all the Security Hidden for users to be easily viewed without having to explore the Menus tree view on the Enhanced Security window.

Changes in the window are made immediately and do not need the OK Button to be clicked.




Below is a description of the individual fields on the window:

*OK Button*
This button will close the Security Hidden window.

*Redisplay Button*
This button will re-populate the window.

*Delete Button*
This button will permanently remove the Security Hidden records with a marked checkbox.

*Open Button*
Use this button to open the command currently selected.

*Security Button*
Use this button to access the Enhanced Security window.

*Export Button*
This button will allow the resources displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

*Export Mode*
Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

*Print Button*
This button will allow a report of the contents of the Security Hidden window to be printed.

The following is a description of the fields on the window:

*Display Mode*
Use this drop-down list in conjunction with the User ID and Company ID fields to control which records are displayed in the Security Hidden view.

*User ID*
This is the User ID for which security is being displayed.

*Company*
This is the company for which security is being displayed.

*Sort Mode*
Use this drop-down list to control the order that the records are displayed in the Security Hidden view.

*Mark All Button*
This button will mark all lines (or all highlighted lines) in the current Security Hidden view as selected.

*Unmark All Button*
This button will mark all lines (or all highlighted lines) in the current Security Hidden view as unselected.

*The Security Hidden window has an Options Menu which can be used to Refresh Application Navigation. This option can be used to update the application's navigation menus to reflect changes made to security without having to exit and re-launch the application.*

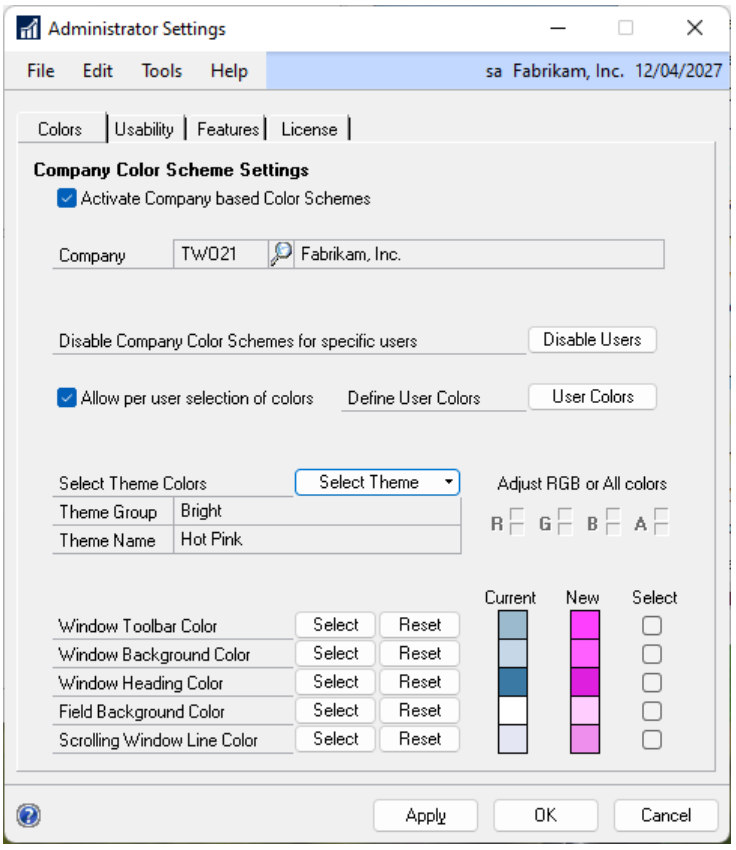
## Administrator Settings

You can open the Administrator Settings window by a selecting Administrator Settings tab from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Administrator Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

The Administrator Settings window can change settings used within GP Power Tools. It is divided into three tabbed sections.

### Colors Tab

The Colors tab contains settings for controlling the Company Color Scheme settings. These color themes are designed to prevent the accidental entry of data into an incorrect company. It supports different colors for each company and, if Binary Stream's Multi-Entity Management is installed, for each entity. There is also the option to have per user override settings for vision impaired and color-blind users.




The following is a description of the individual fields on the window:

*Activate Company based Color Schemes*

This option can be used to change the background colors for each company to allow an effective visual cue as to the company currently being used.

Once the option is activated, the colors for the Window Toolbar Color, Window Background Color, Window Heading Color, Field Background Color and Scrolling Window Line Color can be selected using the Select Theme button drop-down list which offers 110 preset themes. Any Custom Color Themes loaded or created will also be shown in the Select Theme button drop-down list. If a preset or custom theme is selected, the Theme Group and Theme Name will be displayed.

The colors can also be manually selected with the Select Buttons. The Reset Buttons can be used to restore the default colors. You can also use the checkboxes against each color to select which colors will be affected by the Spinner Controls. These up and down buttons can be used to adjust the individual Red, Green and Blue components of the selected colors or to adjust all three values together to lighten or darken the selected colors.


The current color scheme and the new color scheme are displayed. When the Apply Button or OK Button is clicked, the new color scheme will be applied.

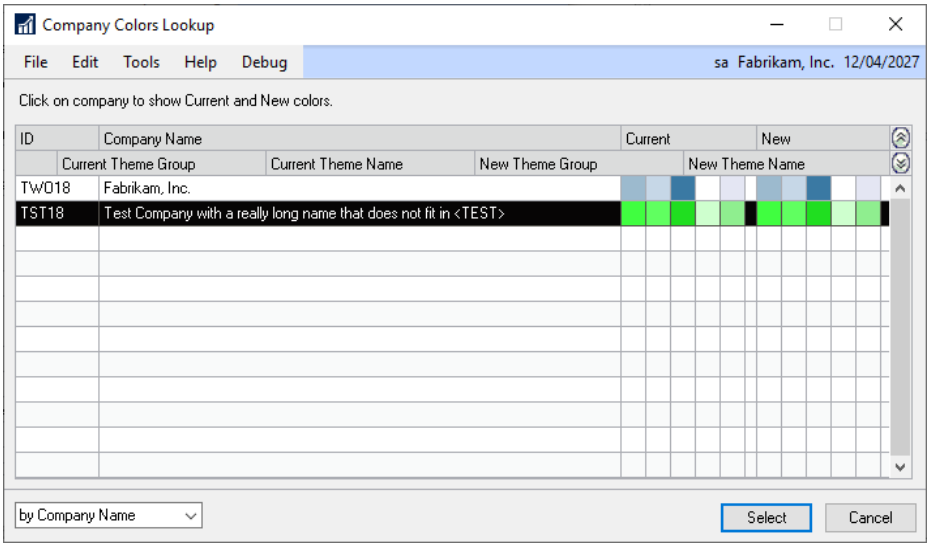
The Color Scheme data is stored at both the Company and System level. This design allows for live company databases to be copied into a test company database while maintaining the correct color schemes.

*When running on the Web Client, the Activate Company based Color Schemes option is disabled as it is not supported.*

### *Company Colors Lookup*

This lookup button can be used to change the company for which the Color Scheme is being edited. When the button is clicked, the Company Colors Lookup window is opened.

The colors for a company are displayed on the window when the company is highlighted. This allows you to see the colors for each company without having to switch companies and open the Administrator Settings window in each company.

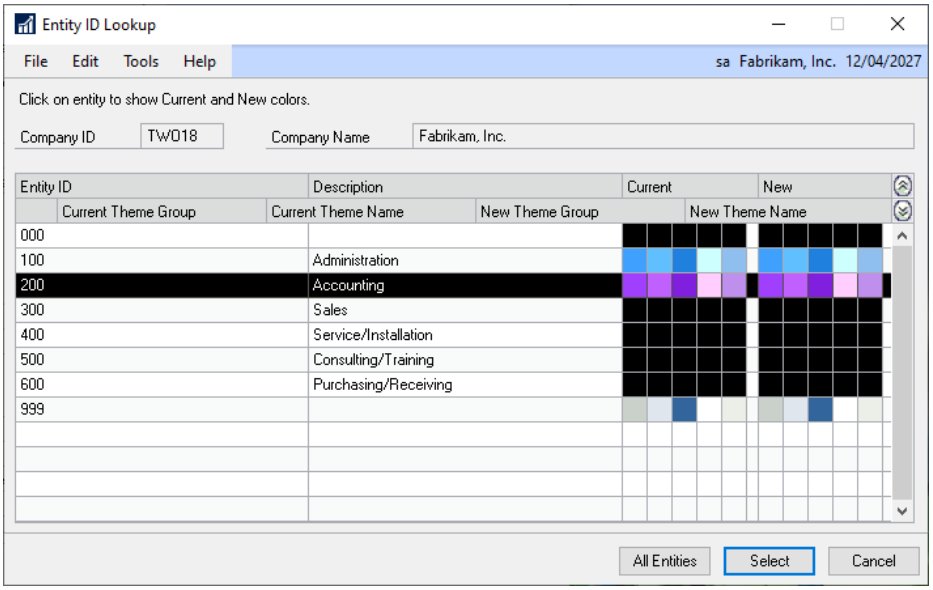



*Changing the Color Schemes for other companies updates the settings in the system database only. This system setting is then transferred into the company database the next time the company is used.*

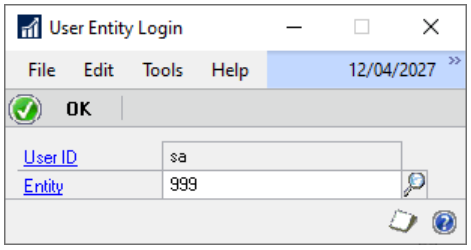
### *Entity ID Lookup*

When Binary Stream's Multi-Entity Management is installed, this lookup button can be used to change the entity for which the Color Scheme is being edited. When the button is clicked, the Entity ID Lookup window is opened to show the entities available in the current company.

The colors for an entity are displayed on the window when the entity is highlighted. This allows you to see the colors for each entity without having to change default entity.



*The color scheme used for entities is only for the default or current entity as configured using the User Entity Login window. If allowing the entity to be changed on a per transaction basis, the color scheme will still be based on the default entity and not the transaction level entity.*

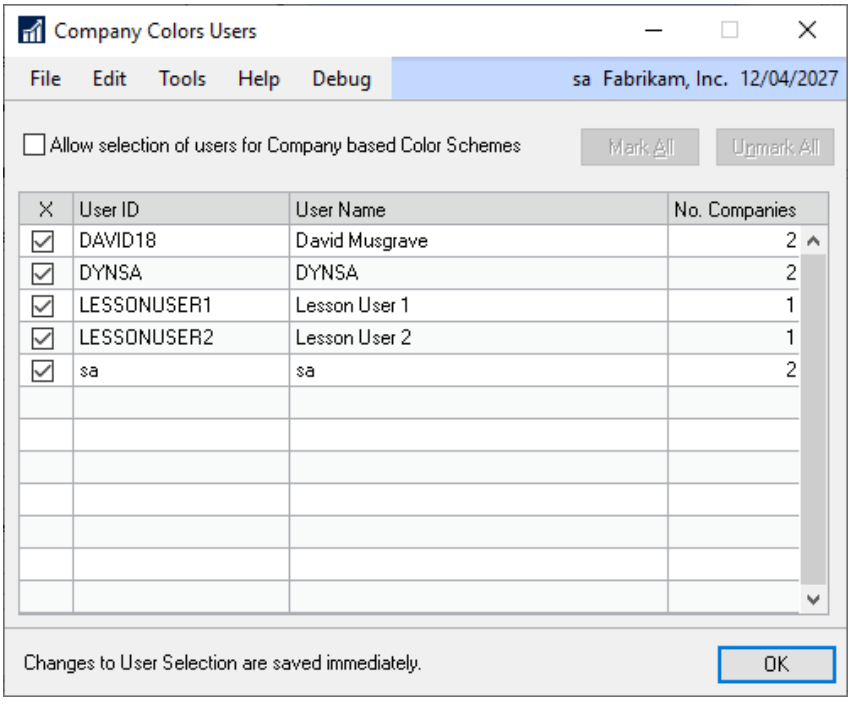



### *Company Colors Users*

This button can be used to optionally control for which users the company-based Color Schemes are enabled. When the button is clicked, the Company Colors User window is opened.

When the Allow selection of users for Company based Schemes checkbox is selected, you will able to use the User Selection checkbox and the Mark All and Unmark All buttons to control for which users the Company based Color Schemes are enabled. By default, all users (including new users) are enabled.

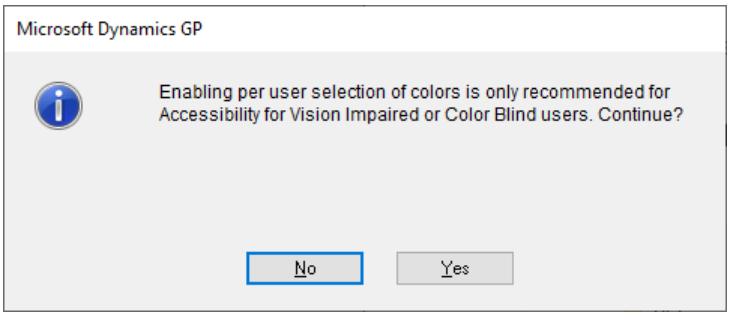
*For each user, the number of companies they have access to is shown, this information can be used when deciding if a user should have the Company based Color Schemes enabled.*




*Changes made to the User Selection checkbox are saved immediately and so will still take effect even if the Administrator Settings window is closed without saving.*

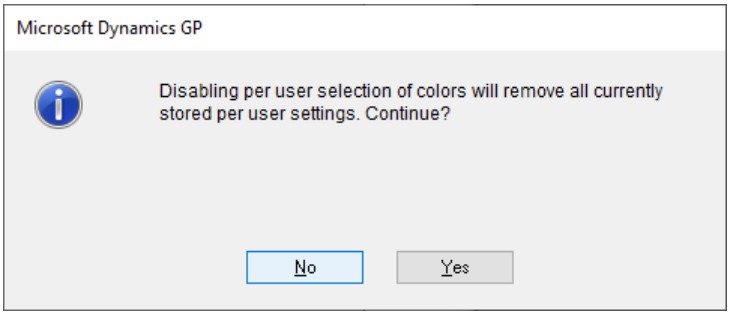
*Allow per user selection of colors*

Use this checkbox to enable and disable Per User Color Selection. This feature is designed to be used for vision impaired or color-blind users to override standard company color scheme. When enabling the following dialog is displayed.



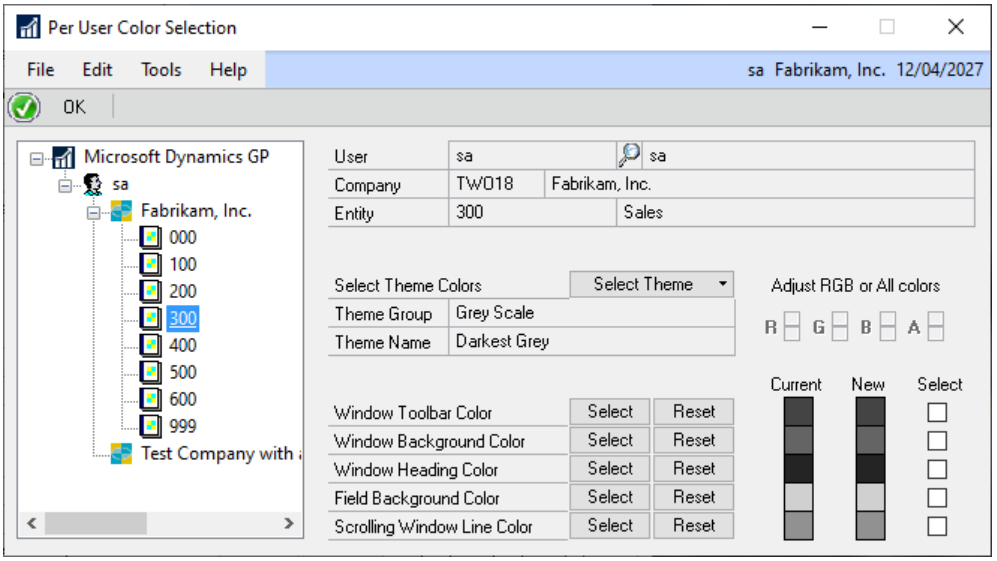

*When using per user selection of colors, using the colors to identify the current company might not work. Keeping the main background color the same as the standard company colors would help avoid this issue.*

When disabling the feature, the following dialog is displayed to explain that any stored per user settings will be removed.



### *User Colors Button*

Click this button to access the Per User Color Selection window.

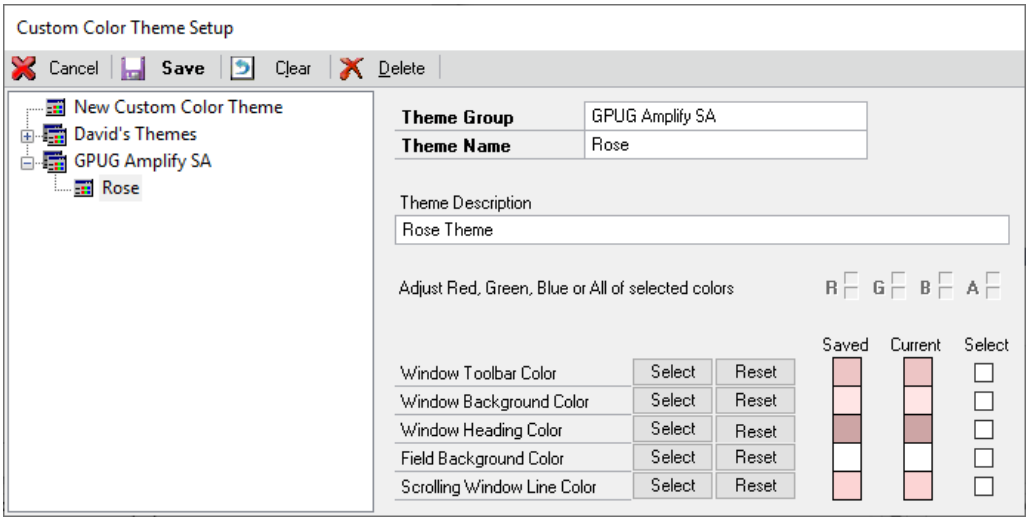



From this window select a user with the lookup to add the user to the tree, then select the user for an "all companies" override, or select individual companies (or entities) to define colors for that company (or entity).

*A High Contrast black and white theme is available from the Accessibility theme group if desired. It makes all the background white so that the text and other window objects stand out.*

### *Custom Color Themes*

Selecting Custom Color Themes from the Select Theme button drop-down list will open the Custom Color Theme Setup window.



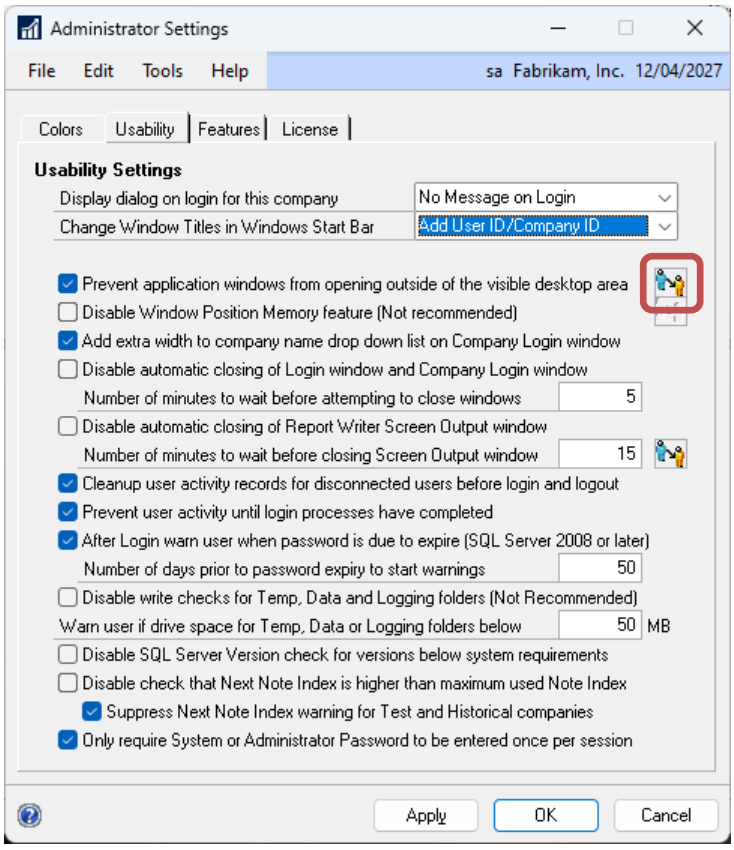

This window can be used to maintain Custom Color Themes. You can create new themes based on the colors Administrator Settings window or create new themes.

The Theme Group and Theme Name fields are required and dictate how the themes are displayed. Themes with the same Theme Group will be grouped together.

Theme colors can be edited with same controls as the Administrator Setting window and can be saved and deleted. Exit the Custom Color Theme Setup window using the Cancel button.

## Usability Tab

The Usability tab contains settings for improving the usability of Microsoft Dynamics GP.




The following is a description of the individual fields on the window:

*Display dialog on login for this company*

This setting can be used to display a dialog after a user has logged into a company. The settings are **No Message on Login** or one of the two choices below:

**Test Company Message**: "This company is set up for testing only. Do not use this company when processing live data."

**Historical Company Message**: "This company is used for storing historical information only. Do not use this company when processing current-year data."

This feature is providing a user interface to the existing dialog functionality as described in Knowledge Base (KB) article 885542:

http://support.microsoft.com/kb/885542

This feature takes effect on the next login or after switching company.

*GP Power Tools resolves the issue when companies marked as Test or Historical with their Company Name containing <TEST> or <HISTORICAL> do not display the appropriate warning dialog on login on a non-English system.*

### *Change Window Titles in Windows Start Bar*

This setting can be used to prefix the window titles as seen in the Windows Start Bar with the User ID or User Name and/or Company ID or Company Name.

This feature is useful when running multiple instances of Microsoft Dynamics GP on a single workstation. It allows users to easily identify which window belongs to each instance of the application by displaying the User ID or User Name and/or Company ID or Company Name in the Windows Start Bar.

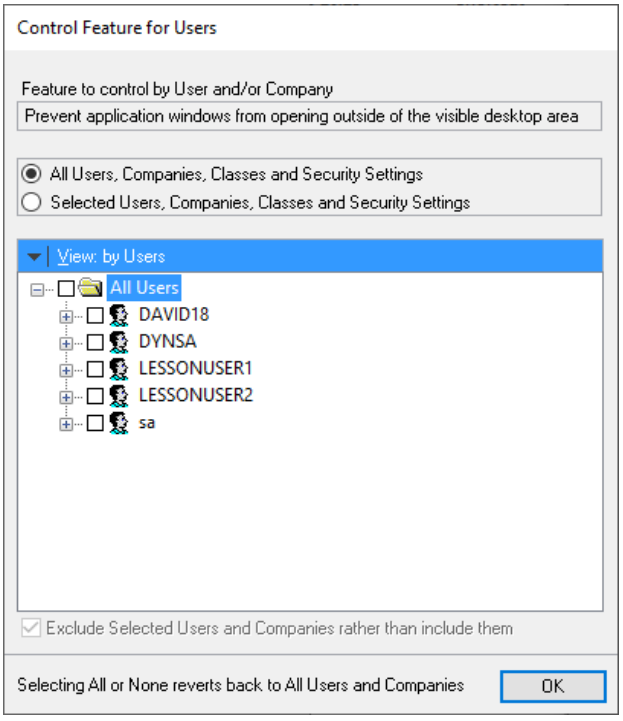
This feature takes effect on next login or after switching company.

*When running on the Web Client, the Change Window Titles in Windows Start Bar option is disabled as it is not supported.*

### *Prevent application windows from opening outside of the visible desktop area*

This setting checks the location of all windows as they open and if they will not be in the visible desktop, their position will be adjusted to make sure they are fully visible.

When this checkbox is selected, you can use the Users Button  to open a window to allow selection of the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this feature to. A red visual cue will be displayed if there are user settings present.



This feature takes effect on next login or after switching company.

*The Window Position Memory window can be used to disable specific windows from this feature if they are hidden windows which are now being displayed when they should remain hidden.*

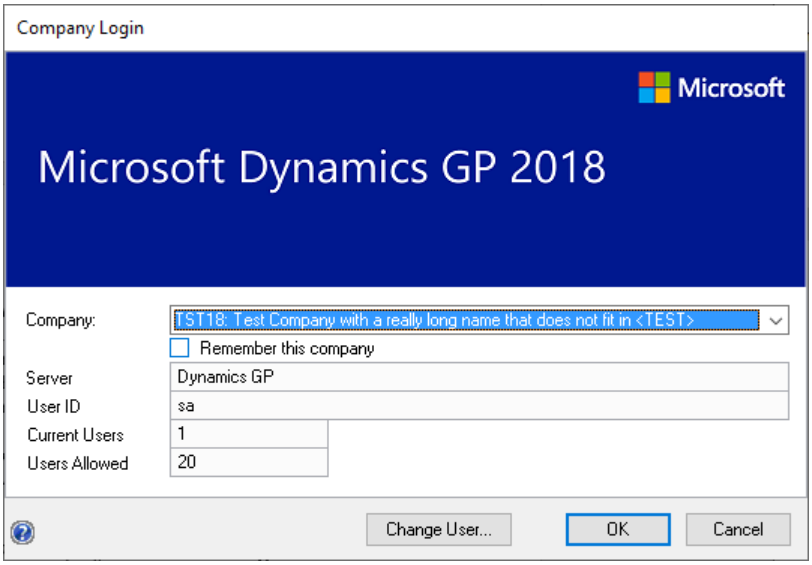
### *Disable Window Position Memory feature*

When this checkbox is selected, the Window Position Memory feature will be disabled. Takes effect immediately for current workstation and on next login for other workstations.

When this checkbox is selected, you can use the Users Button  to open a window to allow selection of the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this feature to. A red visual cue will be displayed if there are user settings present.

### *Add extra width to company name drop-down list on Company Login window*

This setting expands the fields on the Company Login window to use the full width of the window to make it easier to read long company names. This feature uses the MBS_Debug_CompanySwitchWidth Dex.ini setting.




This feature takes effect on next login or after switching company.

### *Disable automatic closing of Login window and Company Login window*

By default, GP Power Tools will close the Login window and Company Login window if they have been left open for a long period of time (defaults to 5 minutes). Selecting this checkbox disables this functionality.

*It is recommended to leave this feature enabled with the checkbox unselected so that the Login window and Company Login windows close automatically when they have been accidentally left open. For a new session, this will exit the Microsoft Dynamics GP rather than leaving the application running without a record in the ACTIVITY table. For an existing session, this will return to the application which also allows the Automatic Logout feature to work (if it is enabled).*

*Number of minutes to wait before attempting to close windows*

This setting specifies the number of minutes before GP Power Tools will attempt to close the Login window or Company Login window if they are left open.

*Disable automatic closing of Report Writer Screen Output window*

By default, GP Power Tools will close the Report Writer Screen Output window if it has been left open for a long period of time (defaults to 5 minutes). Selecting this checkbox disables this functionality.

*Closing the Report Writer Screen Output window automatically allows the background processing queue to continue, meaning that background. scheduled and timed processes will not be held up indefinitely by a report that has been left open.*

*Number of minutes to wait before closing Screen Output window*

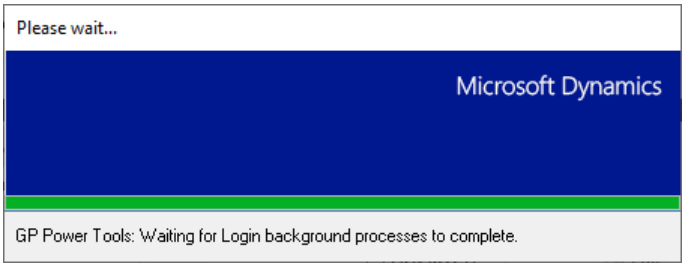
This setting specifies the number of minutes before GP Power Tools will attempt to close the Report Writer Screen Output window if they are left open.

*Cleanup user activity records for disconnected users before login and logout*

When this setting is enabled, GP Power Tools will identify users whose SQL Sessions have been disconnected and cleanup activity records for the disconnected users on login and logout. When this option is enabled, the requirement to remove user activity manually should rarely be required.

*Prevent user activity until login processes have completed*

When this setting is enabled, the modal dialog below will be displayed during the login process until all background processing has been completed. This will prevent a user accessing items on the menu navigation until the cleanup of the menus based on user security has been completed.



*After Login warn user when password is due to expire*

When this setting is enabled, on the first login of the day GP Power Tools will check if the current user's password will expire and if the number of days is less than the specified warning period, a dialog will be displayed offering the user to change their password.

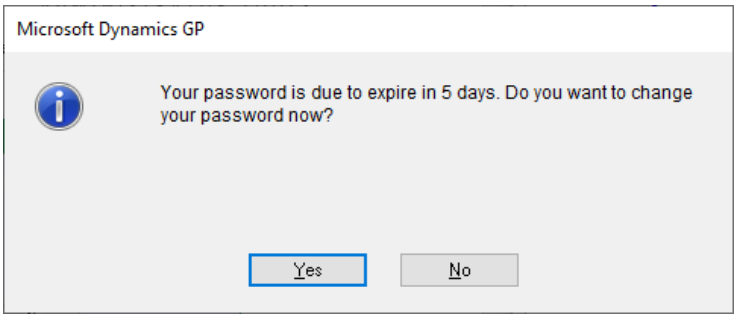
This feature was added to avoid the situation where a user's password could expire at the SQL Server level while they are currently logged into Microsoft Dynamics GP.

*Number of days prior to password expiry to start warnings*

This setting controls the number of days warning for the password expiry warning.

After logging in, if the user's password is going to expire within the selected number of warning days, the following dialog will be displayed.

Selecting Yes will automatically open the User Preferences Password window to allow the user to change their password.



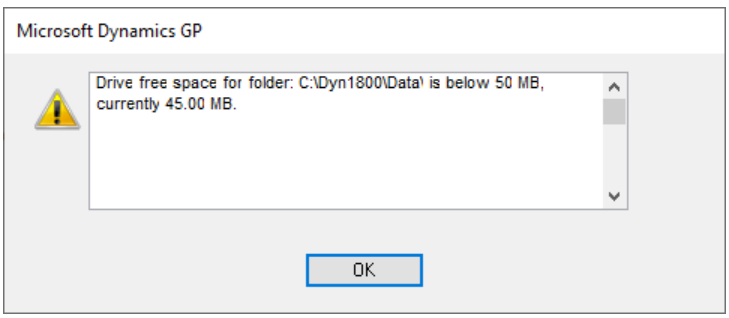

This feature only works for SQL Server 2008 systems or later.

*Disable write checks for Temp, Data and Logging folders*

By default, GP Power Tools will check that write access is available for these three folders. Selecting this checkbox disables this functionality.

*Warn user if drive space for Temp, Data or Logging folders below*

Use this setting to specify how much available space is the minimum value before the system starts to warn users on login. The default is 50MB. If the drive space for one of these three folders falls below the threshold value, a dialog will be displayed during login.

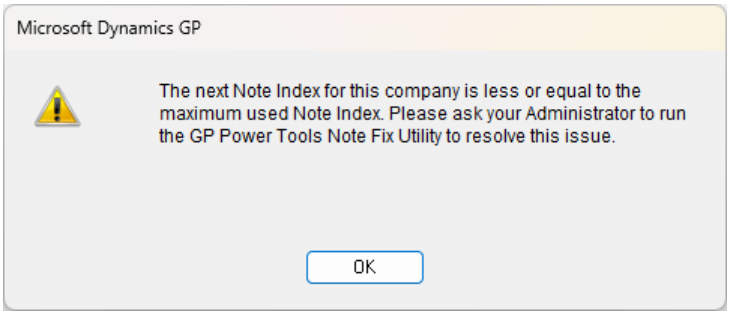


*Disable SQL Server Version check for versions before system requirements*

Use this setting to disable the check on login that the SQL Server version in use meets the minimum supported version as described at the bottom of the System Requirements for Microsoft Dynamics GP article.

*Disable check that Next Note Index is higher than maximum used Note Index*

Use this setting to disable the check on login that the Company's Next Note Index is higher than the maximum used Note Index in the Record Note Master (SY03900) table. This is a quick check that there might be bigger problems with Note Indexes. Use the Note Fix Utility in the Database Tools module to fix issues with Note Indexes.




*If the Next Note Index warning dialog is not displayed when logging into a company, it is not an indication that there are no Note Index issues for that company. It is purely an indication that this very specific quick check did not find the issue it was looking for. The issue being checked is that the Next Note Index for the current company stored in the Company Master (SY01500) table in the system database is less than the maximum note index used in the Record Note Master (SY03900) table in the company database.*

*Suppress Next Note Index warning for Test and Historical Companies*

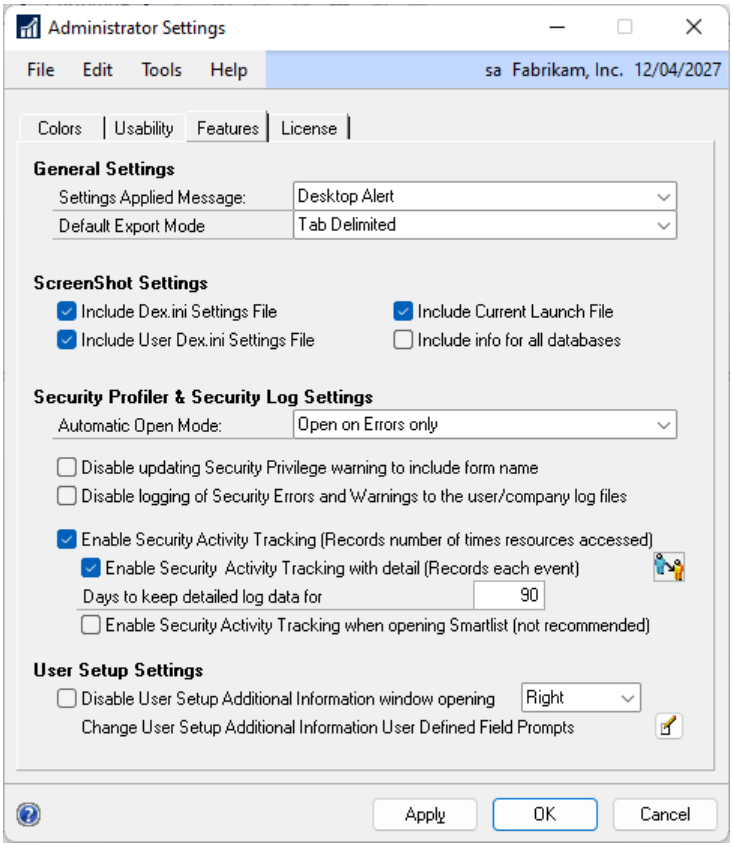
Use this setting to suppress the above warning for Test and Historical Companies. The next Note Index in the Company Master table in the System Database is usually incorrect after a live company database has been restored into a test company database. This option will stop GP Power Tools warning about this issue.

*Only require System or Administrator Password to be entered once per session*

When this setting is enabled and you are asked for the System Password or GP Power Tools Administrator Password, a correct answer will be remembered for the rest of the session and you will not be required to enter the password until you log off or change company.

## Features Tab

The Features tab contains settings for adjusting the behavior of various features of GP Power Tools.




The following is a description of the individual fields on the window:

*Settings Applied Message*

This drop-down list allows the selection of how the "Settings Applied" message should be displayed when click the Apply button on the various settings windows in GP Power Tools.

*Default Export Mode*

Use this drop-down list to select the default Export Mode to be used on windows which support exporting of data.

*Include Dex.ini Settings File*

This checkbox specifies the default setting for ScreenShot.

*Include User Dex.ini Settings File*

This checkbox specifies the default setting for ScreenShot.

*Include Current Launch File*

This checkbox specifies the default setting for ScreenShot.

*Include info for all databases*

This checkbox specifies the default setting for ScreenShot.

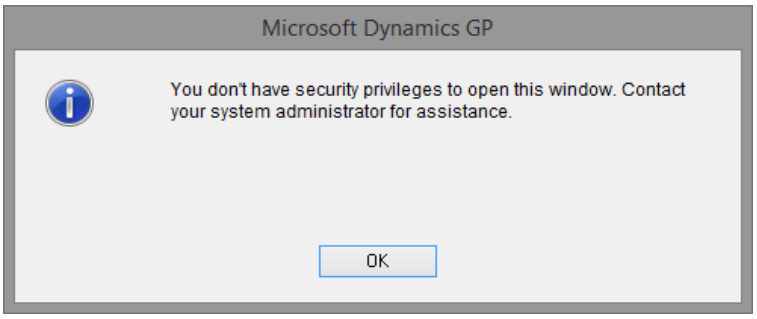
### *Automatic Open Mode*

This option controls whether the Security Profiler window should automatically open when a security error occurs. The options are Do not open Automatically, Open on Errors only; and Open on Error & Warnings.

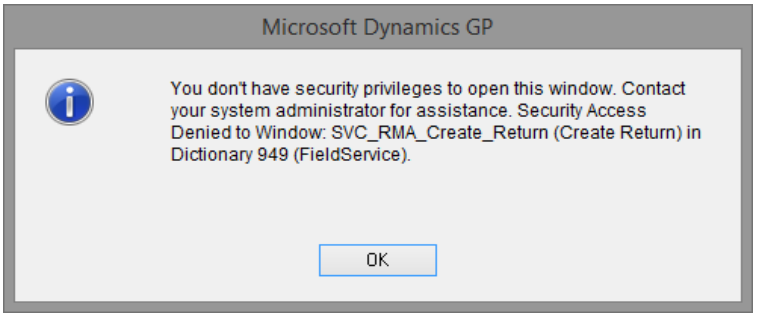
*An Error is a situation that will cause the application to generate a dictionary not loaded or permission denied error dialog. A Warning is a situation where no error dialog will be generated, but the resources defined in the settings will not be opened as expected.*

### *Disable updating Security Privilege warning to include form name*

This option controls whether the Security Privilege warning dialog (screenshot below) includes additional information about the resource for which security access has been denied.




By default, once GP Power Tools is installed, additional information will be included on the dialog (screenshot below). This will help administrators identify the issues even if the Security Profiler window is not in use.




Enabling this option will disable the additional information and revert back to the standard dialog.

### *Disable logging of Security Errors and Warnings*

This option disabled the logging of Security Errors and Warnings to the GPPTools_<User>_<Company>.log log files.

*Enable Security Activity Tracking*

This option enables the recording of statistics for all Security events to track the number of times resources are accessed and whether the access for granted or denied. Use the Security Log window to review the data captured.

When saving after this option has been disabled, if the logs contain data, a dialog will be displayed asking if you wish to remove the data.

*Enable Security Activity Tracking with detail*

This option enables the recording of details of all Security events to track each time resources are accessed and whether the access for granted or denied. Use the Security Log window to review the detailed data captured.

*This feature can be controlled on a per user and/or company or by User Class, Security Role, Security Task or Security Modified Alternate ID. This will allow the amount of detailed data captured to be restricted if desired.*

When saving after this option has been disabled, if the logs contain data, a dialog will be displayed asking if you wish to remove the data.

*Days to keep detailed log data for*

This option specifies how many days to keep detailed logging data for before it is automatically removed.

*Enable Security Activity Tracking when opening Smartlist*

This option re-enables the recording of Security events when opening the Smartlist window. By default, capturing this information has been disabled as it can cause performance issues especially when your system has a large number of Smartlist favorites.

*Disable User Setup Additional Information window automatically opening*

When this checkbox is selected, the User Setup Additional Information window will not open automatically when the User Setup window is opened. The position of the window when it opens, Below or to the Right of the User Setup window can be selected.

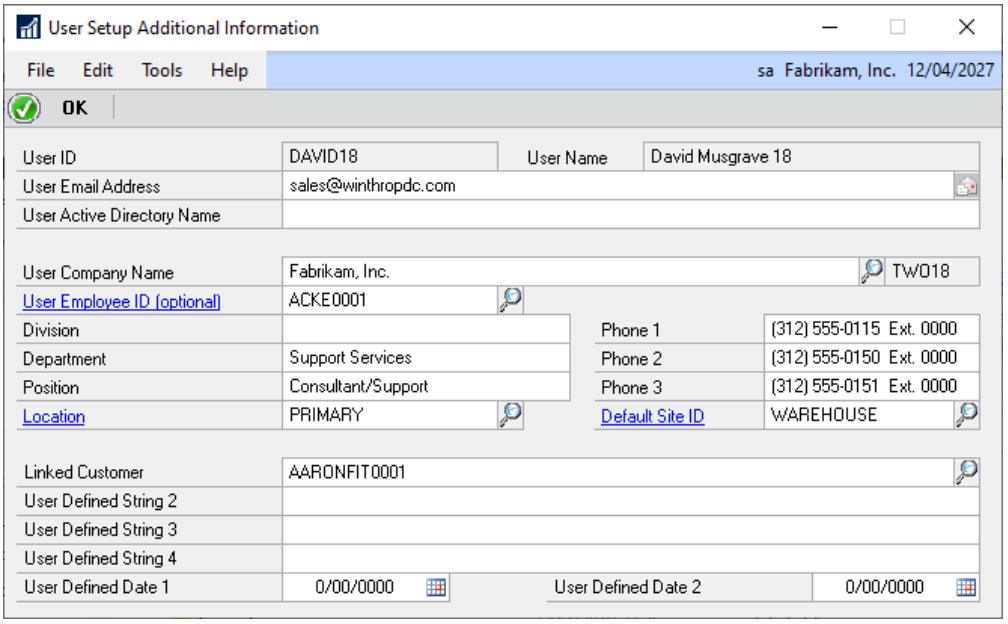
This window is used store additional information against each Microsoft Dynamics GP user. The User Email Address is used by the Database Validation feature to send emails when resetting users' passwords.

If a default company is selected and the system is currently logged into that company, the lookups can be used to select Employee ID and related information.

The Default Site ID field is used to auto populate the Default Site on the following windows:

- Item Transaction Entry
- Item Transfer Entry
- Item Enquiry
- Sales Transaction Entry
- Invoice Entry
- Purchase Requisition Entry
- Purchase Order Entry
- Receivings Transaction Entry

There are also four User Defined String fields and two User Defined Date fields. The prompts for these six fields can be defined using the following option on the Administrator Settings window.

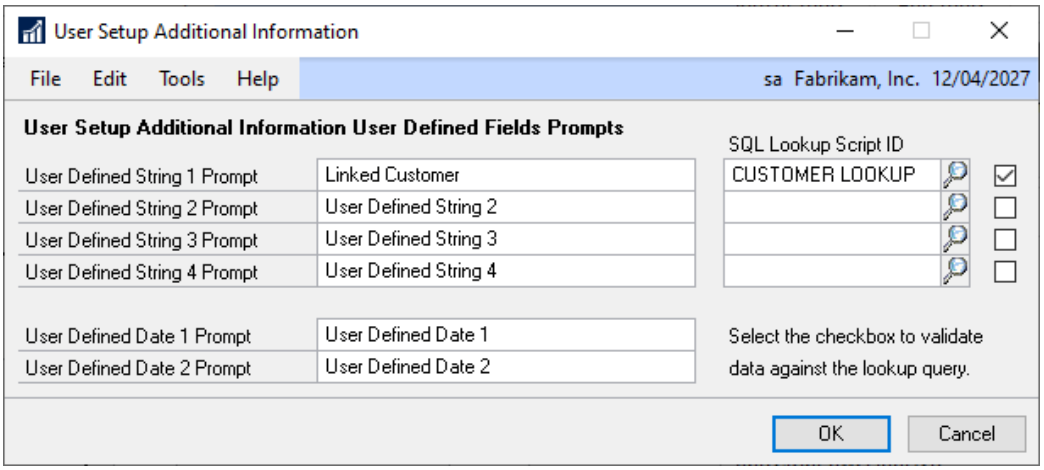



The User Setup Additional Information window is synchronized with the User Setup window so the information is displayed, saved and deleted using the controls on the User Setup window. Clicking the OK Button on the window, just closes the window, but does not save anything until the user is saved.

*Even if the User Setup Additional Information window does not open automatically, it can be opened using the Additional Menu on the User Setup window or pressing the Ctrl-I keyboard shortcut.*

### *Change User Setup Additional Information User Defined Field Prompts*

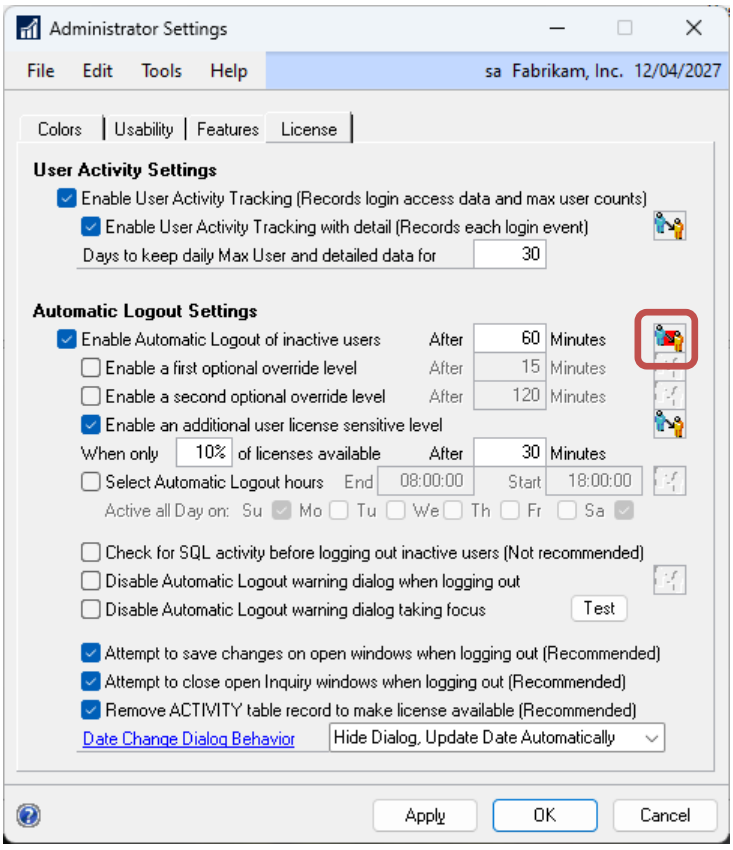
Click the Edit Button to open the window to change the prompts for the User Defined String fields and User Defined Date fields used on the User Setup Additional Information window.




*If the Developer Tools module is registered, the string user defined fields can be linked to lookups and optionally validated against the rows in the lookup to ensure the record entered does exist. This uses the SQL Lookup feature which is usually used on the Parameter List Maintenance window.*

## License Tab

The License tab contains settings for enabling and disabling License Management features of GP Power Tools.




The following is a description of the individual fields on the window:

### *Enable User Activity Tracking*

When this checkbox is selected, the User Activity Log feature will be enabled. Takes effect immediately for current workstation and on next login for other workstations.

The User Activity Log tracks statistics when users' login and logout and tracks the daily maximum session count on a system, user and company basis.

When saving after this option has been disabled, if the logs contain data, a dialog will be displayed asking if you wish to remove the data.

### *Enable User Activity Tracking with detail*

This option enables the recording of detailed logs of each login and logout event. Use the User Activity Log window to review the detailed data captured.

*This feature can be controlled on a per user and/or company or by User Class, Security Role, Security Task or Security Modified Alternate ID. This will allow the amount of detailed data captured to be restricted if desired. A red visual cue will be displayed if there are user settings present.*

When saving after this option has been disabled, if the logs contain data, a dialog will be displayed asking if you wish to remove the data.

*Days to keep daily Max User and detailed data for*

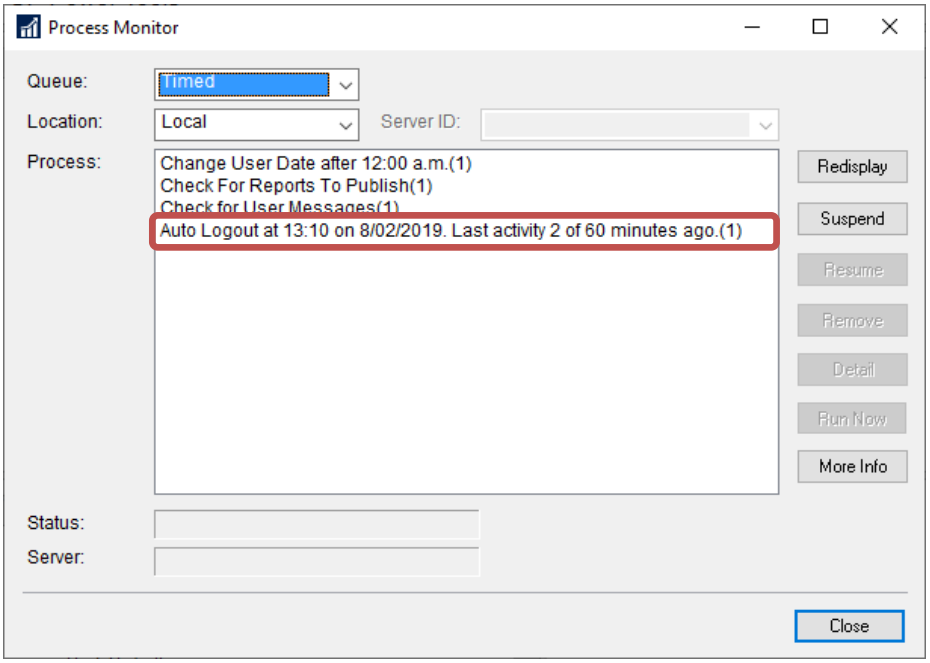
Use this setting to control for how many days the daily maximum session count data and detailed logging data are kept. This data includes a list of the sessions logged into Microsoft Dynamics GP when the maximum count was reached.

*Enable Automatic Logout of inactive users*

When this checkbox is selected, the Automatic Logout feature will be enabled. This feature takes effect immediately for current workstation and on next login for other workstations.

Automatic Logout mimics the user selecting File >> Exit from the menus and so will perform a well-behaved exit of Microsoft Dynamics GP. It does NOT do anything that will force terminate the application, as this is dangerous and can cause orphaned data and data corruption.

Automatic Logout uses timed background process which executes every minute and will attempt to logout once the user has been inactive for more than the number of minutes specified. You can look at the Process Monitor to see the process and its current status.




Use the After X Minutes field to define how long a user must be inactive before Automatic Logout attempts to log out of Microsoft Dynamics GP.

When this checkbox is selected, you can use the Users Button  to open a window to allow selection of the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this feature to. A red visual cue will be displayed if there are user settings present.

*To facilitate the smooth running of the Automatic Logout functionality when the Send Users Message feature of Microsoft Dynamics GP is used, GP Power Tools replaces the system dialog popup windows with custom forms. This behavior can be controlled with the MBS_Debug_UserMessageReplace Dex.ini setting.*

### *Enable a first optional override level*

When this checkbox is selected, you can enter an optional shorter time limit for Automatic Logout.

Use the Users Button  to select the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this shorter time level to.

### *Enable a second optional override level*

When this checkbox is selected, you can enter an optional longer time limit for Automatic Logout.

Use the Users Button  to select the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this longer time level to. A red visual cue will be displayed if there are user settings present.

### *Enable an additional user license sensitive level*

When this checkbox is selected, the Automatic Logout feature will check the number of available licenses remaining and once it reaches the specified threshold, Automatic Logout will use a shorter time before attempting to log out. This allows the feature to get more aggressive with logging out inactive users when the available licenses are low.

Use the When only X% of licenses available field to define at what percentage of available licenses remaining does the second level of Automatic Logout activate.

Use the After X Minutes field to define how long a user must be inactive before the second level of Automatic Logout attempts to log out of Microsoft Dynamics GP.

### *Select Automatic Logout hours*

When this checkbox is selected, you can enter business hours and work days during Automatic Logout will not be active. Using this option will not help limit user license usage during the day, but will help ensure users are logged out outside of business hours.

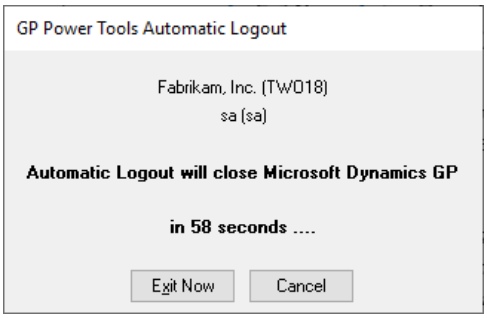
Use the Users Button  to select the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this feature to. A red visual cue will be displayed if there are user settings present.

*Check for SQL activity before logging out inactive users*

When this checkbox is selected, the Automatic Logout feature will check for the last activity at the SQL Server level before logging out. Enabling this option is not recommended as other background timed processes can update the last SQL activity and thus prevent Automatic Logout from working. There is a warning displayed when selecting the option.

*Disable Automatic Logout warning dialog when logging out*

When this checkbox is selected, the Automatic Logout feature will disable the Automatic Logout Warning Dialog window which counts down the final minute and gives the user the choice to Exit Now or Cancel.




When this checkbox is selected, you can use the Users Button  to open a window to allow selection of the users and/or companies, or User Class, Security Role, Security Task or Security Modified Alternate ID to apply this feature to. A red visual cue will be displayed if there are user settings present.

*If the User Activity Log feature is enabled, statistics are tracked on how many times a user cancels this dialog as well as how many times the Automatic Logout feature triggers and how many of those times resulted in a successful attempt to logout.*

*Disable Automatic Logout warning dialog taking focus*

When this checkbox is selected, the Automatic Logout feature will disable the Automatic Logout Warning Dialog window attempting to take focus and jump to the foreground when it refreshes each second of the countdown. You can use the Test Button to open the dialog on request to test the behavior of the dialog.

*Please note that if you click Exit Now on the dialog, even when opened with the Test Button, Automatic Logout will attempt to exit Microsoft Dynamics GP.*

*Attempt to save changes on open windows when logging out*

When this checkbox is selected, the Automatic Logout feature will attempt to close open windows in reverse order to when they were opened. If a window needs saving and has all required fields entered, Automatic Logout will simulate the user selecting the Save Button or OK Button.

This additional "smart" option will increase the chances of a successful log out even if a user leaves their screen with unsaved data while ensuring that their changes are not lost.

*Attempt to close open Inquiry windows when logging out*

When this checkbox is selected, the Automatic Logout feature will attempt to close open Inquiry windows by clearing them.

This additional "smart" option will increase the chances of a successful log out even if a user leaves an Inquiry window with invalid data in a field.

*Remove ACTIVITY table record to make license available*

When this checkbox is selected, the Automatic Logout feature will remove the current user's record from the ACTIVITY table. This will allow the user to finish working on the window they were using, but they will be unable to open any new windows.

This additional "smart" option will free up the license to be used by other users while still allowing the current user to complete the task they were working on.

*Please note that if there are windows which cannot be closed or if a dialog opens, the Automatic Logout attempt will be suspended until the dialogs have been handled by manual intervention.*

*Date Change Dialog Behavior*

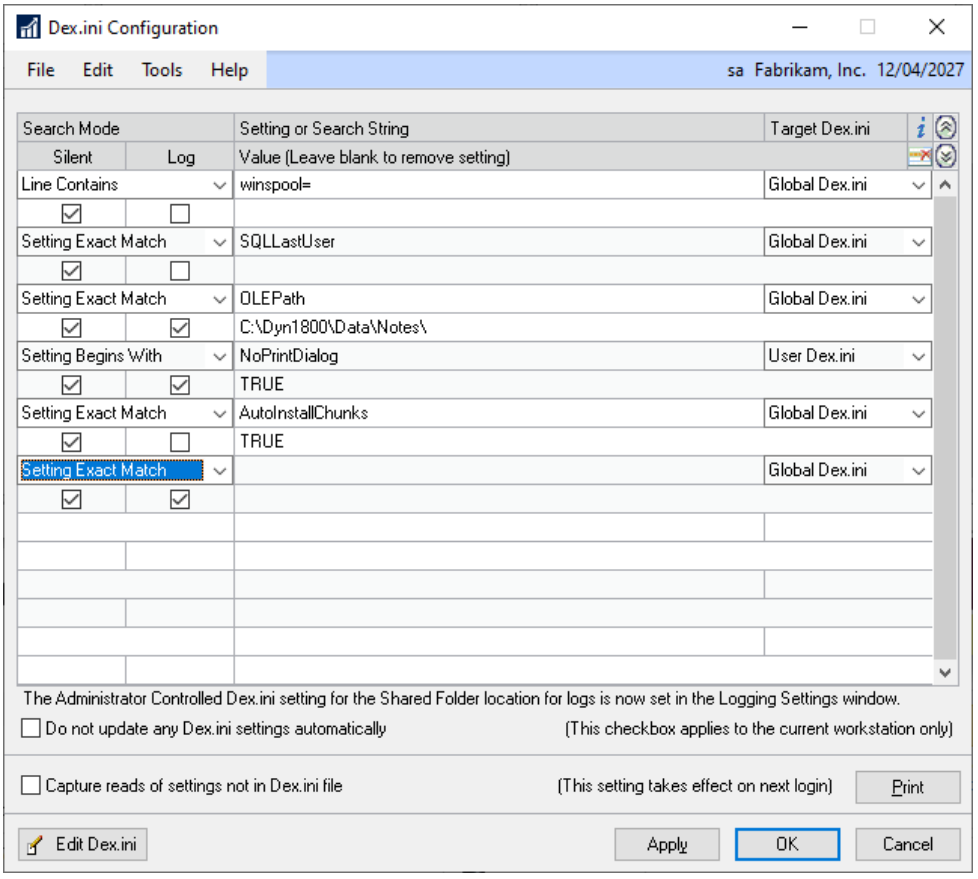
Use this option to override the SuppressChangeDateDialog and SuppressChangeDateForce Dex.ini settings to control the behavior of the Date Change dialog that normally will open at midnight.

*Using the default setting of Hide Dialog, Update Date Automatically will allow the date to change automatically at midnight without opening a dialog. This will avoid the dialog stopping background, scheduled and timed processes being held up while the dialog is open.*

## Dex.ini Configuration

You can open the Dex.ini Configuration window by selecting Dex.ini Configuration from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Dex.ini Configuration from the Options button drop list on the main window. This is an Advanced Mode feature.

The Dex.ini Configuration window can be used to automate changes to Dex.ini settings for all workstations in the system.




The following is a description of the individual fields on the window:

### *Settings List*

This list contains the Dex.ini settings to be checked on login. The setting can be specified with an exact value (this is needed to add a new setting), or can be specified using a "contains" or "begins with" search. The search can be applied against the Dex.ini settings listed in the Dex.ini file (i.e. left of the = sign) or against the entire line in the file. Using a search can find and update multiple lines in the Dex.ini file if more than one setting or line meets the Search Mode and Search String criteria.

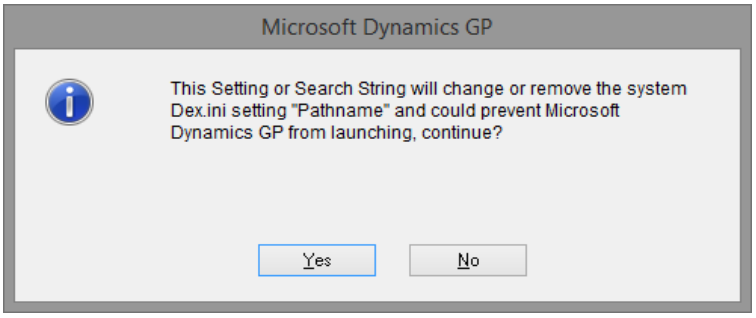
The Silent checkbox should be checked (default) if the Dex.ini setting should be updated without asking the user.

The Log checkbox should be checked (default) if the Dex.ini setting changes should be recorded in the Debugger log files.

The Value field contains the value to change the Dex.ini setting to. Leaving this field blank will remove the Dex.ini setting from the Dex.ini file.

The Target Dex.ini field allows the selection of whether this setting should be applied to the Global Level Dex.ini file (default), to the User level Dex.ini file, or to Both Dex.ini files.

*Before the line in the Setting List is saved, it is checked for possible damaging settings and if they exist an additional confirmation is required. Possible changes to the following Dex.ini settings are detected: Pathname, Initial, Synchronize, Workstation, Workstation2, FileHandler, DatabaseType.*




*Do not update any Dex.ini settings automatically*

This field can be used on individual workstations to prevent GP Power Tools from automatically changing any Dex.ini Settings. It will update the MBS_Debug_ConfigurationOverride Dex.ini setting. This can be useful on test or administration workstations which might not want their Dex.ini settings changed.

*Capture reads of settings not in Dex.ini file*

Select this checkbox to ask GP Power Tools to capture a log of any Dex.ini settings with are read but do not exist in the Dex.ini file. This option can be used to find Dex.ini settings that are undocumented.

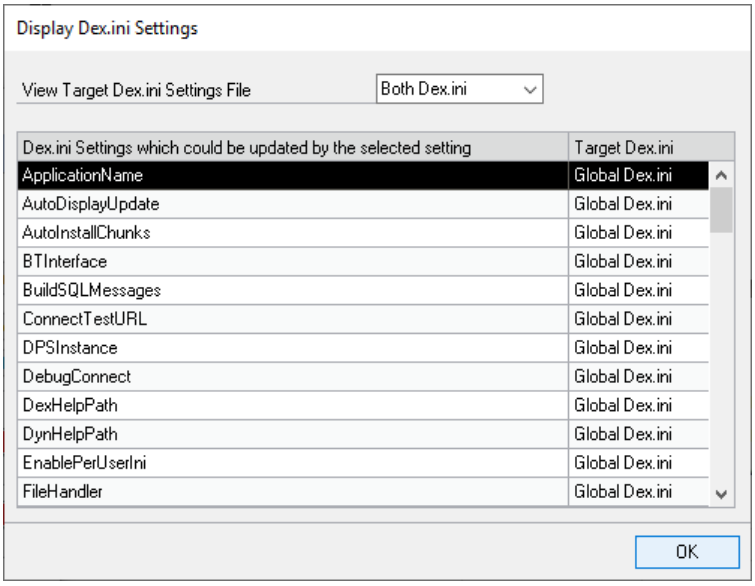
*Care should be taken when using undocumented Dex.ini settings as the effects of the setting cannot be fully known unless access to the source code that references the setting is available.*

*Print Button*

Use this button to print the Dex.ini settings read that do not exist in the Dex.ini file captured.

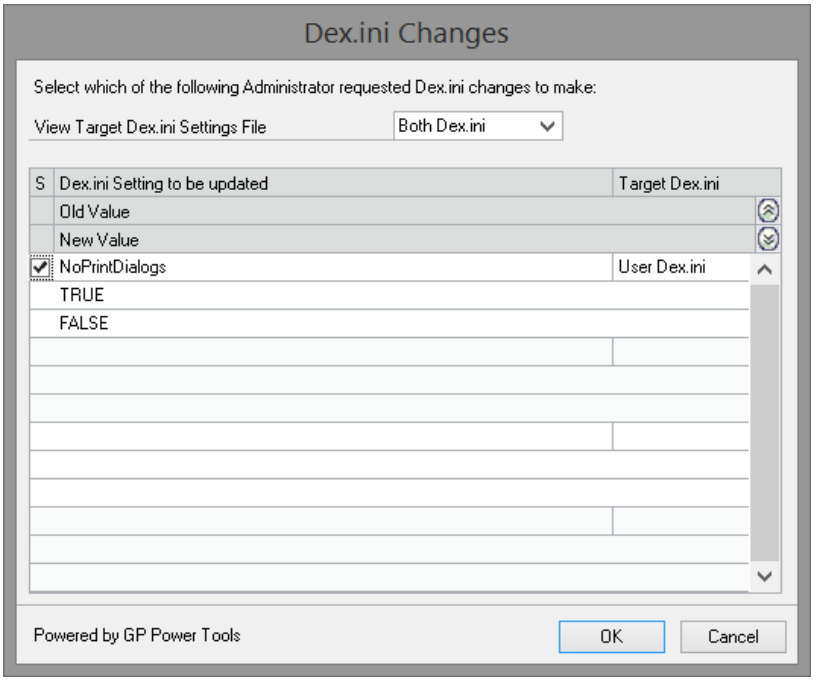
The Apply Button can be used to save the changes to the setup files without closing the window.

Click the Info button to display a list of Dex.ini settings that can be changed by the current Search Mode and Search String criteria. The Display Dex.ini Settings window will open.



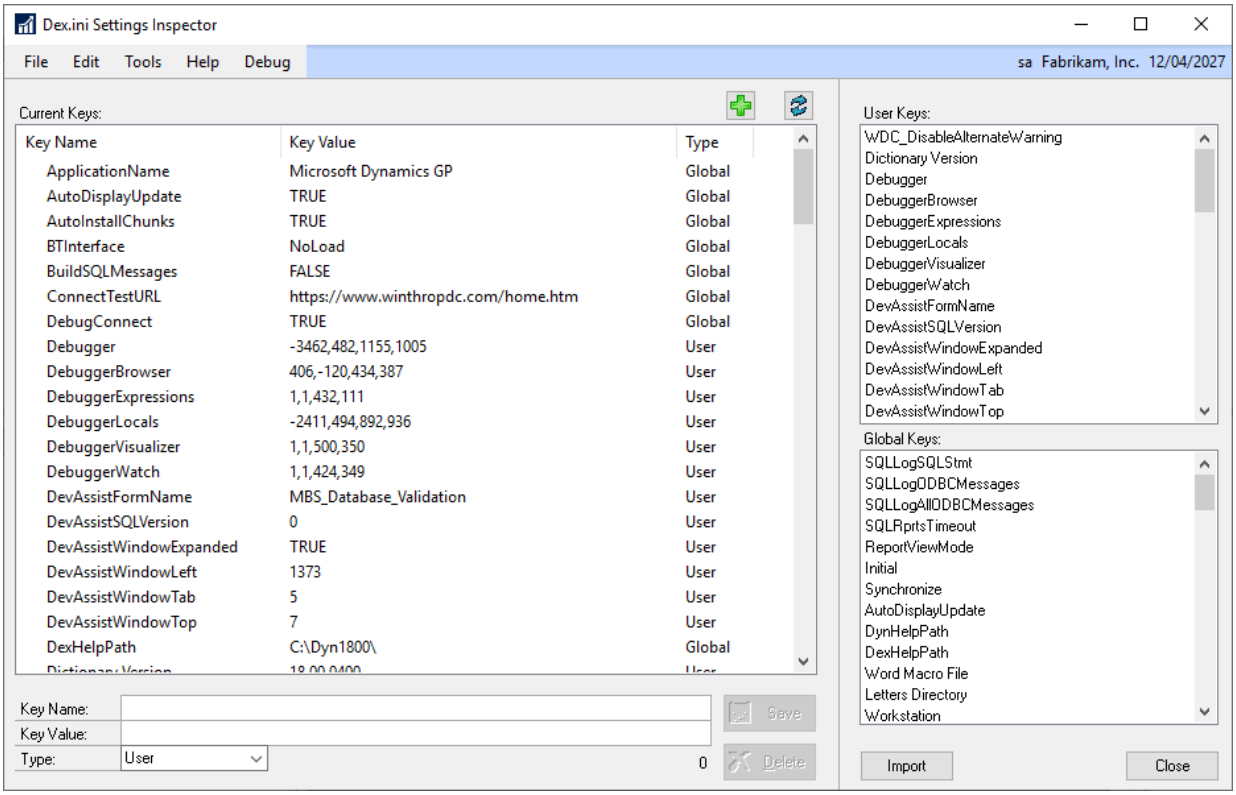

Setting changes specified in the Dex.ini Configuration window are checked against the Dex.ini file when a user logs in. The system looks for settings which differ from the specified values. If the change is marked as Silent, the setting will be automatically updated.

If any changes need to be made where the Silent checkbox is not selected, then the user will be presented with a dialog asking them to confirm which changes should be applied.




*If a user opts to deselect a Dex.ini change that setting will be displayed again when the user logs in again or changes company.*

You can use the Edit Dex.ini button to open the Dex.ini Settings Inspector window. This window allows viewing and editing of both the Global and User level Dex.ini settings.

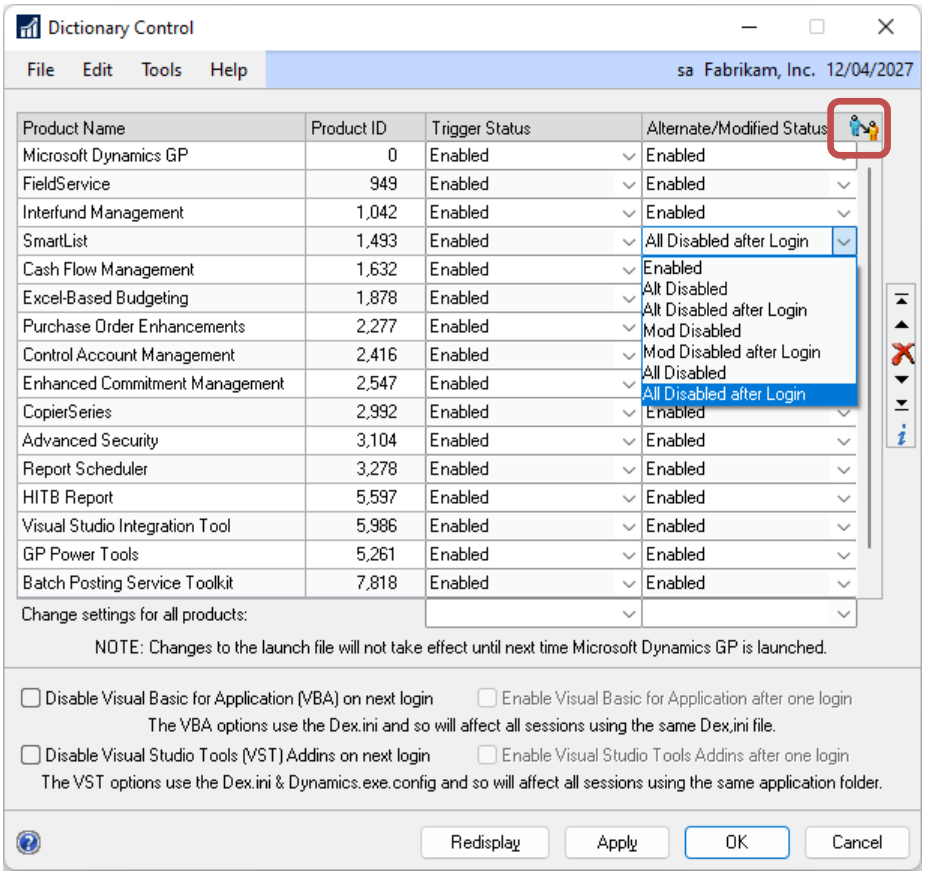


## Dictionary Control

You can open the Dictionary Control window by selecting Dictionary Control from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Dictionary Control from the Options button drop list on the main window. This is an Advanced Mode feature.

Dictionary Control can be used to troubleshoot issues with third party dictionaries. You can effectively remove dictionaries from the system one-by-one until the issue stops. Then the last dictionary to be removed can be investigated further.

You can use Trigger Status to disable Dexterity triggers for a specific Product in a similar fashion to the Customization Status window in Microsoft Dynamics GP. The added benefit of Dictionary Control is that it can remember the settings and automatically disable the product on the next login.

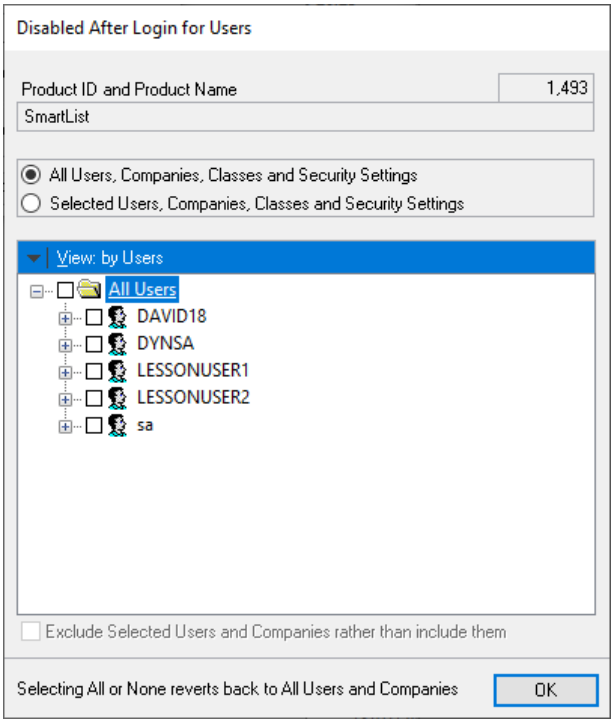



Dictionary Control can also disable alternate and/or modified windows for third party dictionaries using the Alternate Status option. This does not change any security settings.

The drop-down lists at the bottom of the window can be used to change settings for all dictionaries.

*Using Dictionary Control to disable the triggers and alternate windows for a third party dictionary can produce the same effect as removing the dictionary from the Dynamics.set launch file without requiring any backups or manual editing.*

If you set a product to one of the "after login" options, you can use the User button in the top right corner of the window to specify which users and companies should have this dictionary control setting in effect after logging in. Once clicked the Disabled After Login for Users window will open.




You can view this window by Users and/or Companies, User Classes, Security Roles, Security Tasks or Security Modified Alternate IDs and navigate the tree to select the options as required.

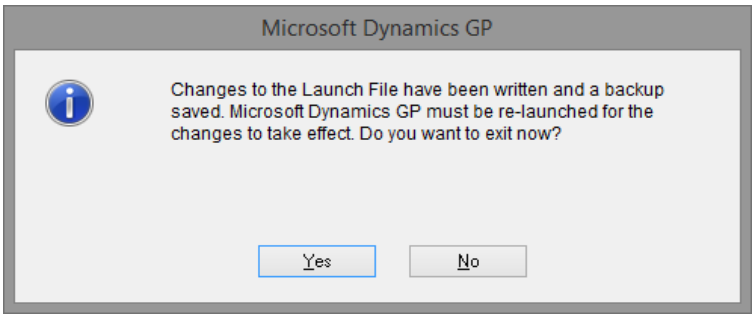
*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the disabling should not take effect.

Sometimes issues can occur because of the order of the dictionaries in the system. Different dictionaries adding triggers for the same event in the application can sometimes clash causing unexpected or undesirable results. The order that triggers from different products will execute is affected by the order of the products in the Dynamics.set Launch File. By changing the order of the products, you can change the order of the triggers and avoid the issues.

*Under most circumstances having two or more dictionaries triggering from the same event would not cause any problems regardless of the order the triggers are executed in. Sometimes, a trigger from one dictionary can make changes to data which affect the behaviour of a trigger from a second dictionary thus causing the code to fail. It is this type of situation which can often be fixed by re-ordering the dictionaries.*

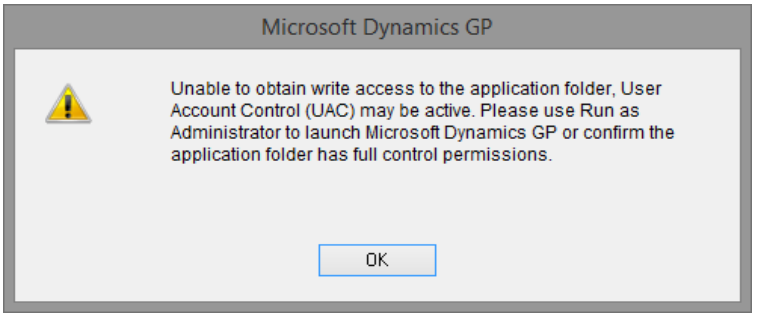
Dictionary Control allows the order of the products to be changed using the Top, Up, Down and Bottom buttons. You can also remove a product with the Delete Button. Any changes to the Dynamics.set launch file will be saved when OK is clicked. You will be requested to restart Microsoft Dynamics GP after the changes have been saved.




A backup of the original file will be saved as Backup X of Dynamics.set, where X will be a number starting at 1.

*If using the Dictionary Control window to disable access to modified windows be aware that by displaying the original window, users might have access to fields previously hidden or disabled on the modified version of the window.*

*If User Account Control (UAC) is preventing write access to the application folder, you will see the following dialog displayed. You will need to use Run as Administrator to allow access and complete the changes.*



Dictionary Control now has the ability to disable Visual Basic for Applications (VBA) and Visual Studio Tools (VSTools) on next login.

The following is a description of the additional checkboxes on the window.

*Disable Visual Basic for Applications (VBA) on next login*
This checkbox disables Visual Basic for Applications (VBA) when the application is restarted. This option uses the VBADisable Dex.ini setting.

*Enable Visual Basic for Applications after one login*
This checkbox automatically re-enables Visual Basic for Applications (VBA) for the application after the first restart. This option uses the MBS_Debug_VBADisableReset Dex.ini setting.

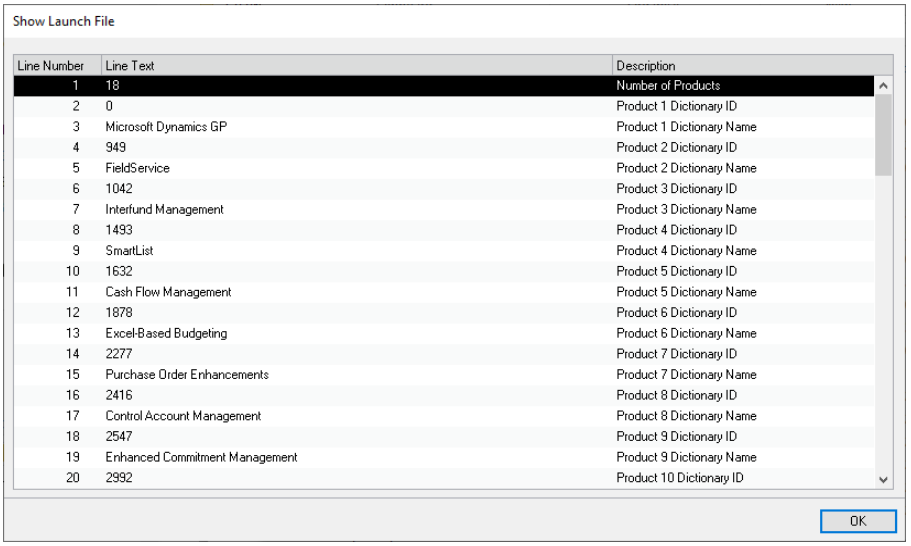
*Disable Visual Studio Tools (VST) Addins on next login*
This checkbox disables Visual Studio Tools (VST) Addins when the application is restarted. This option uses the MBS_Debug_VSTDisable Dex.ini setting.

*Enable Visual Studio Tools Addins after one login*
This checkbox automatically re-enables Visual Studio Tools (VST) Addins for the application after the first restart. This option uses the MBS_Debug_VSTDisableReset Dex.ini setting.

*The Visual Basic for Applications and Visual Studio Tools options are not available if running on the Web Client. The Visual Studio Tools options will be disabled if User Account Control (UAC) is preventing write access to the application folder. This is because the Dynamics.exe.config file must be renamed as part of the process of disabling Visual Studio Tools Addins.*

*Disabling Visual Studio Tools will disable the ability to execute .Net scripts (Visual C# or Visual Basic.Net) as well as the ability to execute Dexterity sanScript in the context of a Modified dictionary.*

If you want to check exactly what is contained in the Dynamics.set launch file and confirm that each line is in the correct position you can click the Info button to open the Show Launch File window.




The Description column in this window describes what information should be on the current line of the file for the file to be valid.

*To use Dictionary Control, a user must have security access to the Customization Status window. This window may have access disabled automatically on each login when using Field Level Security and Field Security IDs are active for the current user and company.*

*Dictionary Control cannot be used to disable alternate windows and forms or triggers in GP Power Tools. As the core Microsoft Dynamics GP dictionary cannot have alternate windows, Dictionary Control cannot be used to disable alternate windows.*

*When running on the Web Client, Dictionary Control cannot be used to modify the launch file and so the movement and delete buttons are disabled.*

Dictionary Control handles checks by third party products of the security tables to ensure they correctly identify which version of a window is currently open. This prevents triggers running on the incorrect version of a window and generating errors.

*To disable SQL Triggers which might be used for customizations, use the SQL Trigger Control window in the Database Tools module.*

## Company Login Filter

You can open the Company Login Filter window by selecting Company Login Filter from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Company Login Filter from the Options button drop list on the main window. This is an Advanced Mode feature.

The Company Login Filter window is used to set up filtering and re-ordering of the companies available in the Company Login window for a user depending on the instance of the application environment currently being used. The companies available for a specific instance of the application environment can be controlled by a Dex.ini Setting which selects the active Company Login Filter profile and optionally by path of the Launch File used to start Microsoft Dynamics GP (usually Dynamics.set).

Examples of Use:

- On a multinational system, where different countries or regions have different localization dictionaries (such as VAT or GST), you can ensure that a company database is only used with the matching application client for each company.

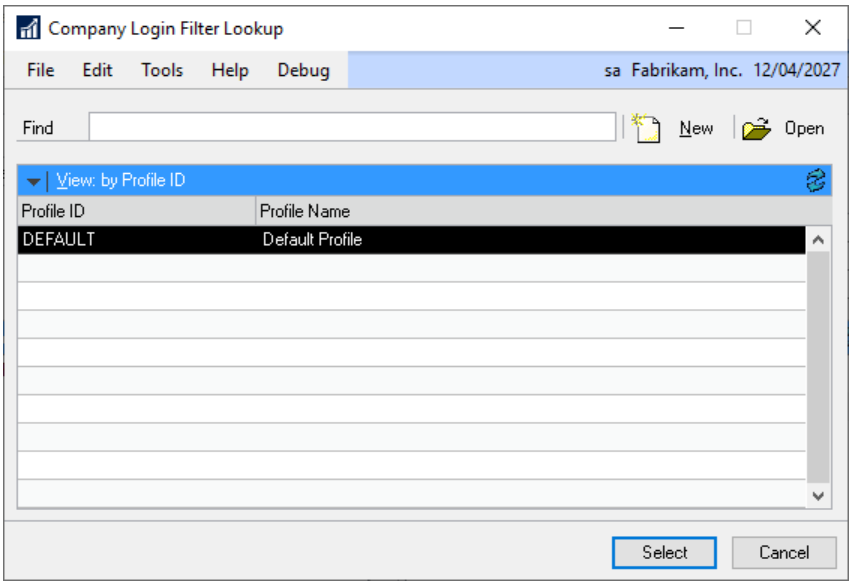
- On a system with different customization dictionaries or different custom forms and reports for different companies, you can ensure that the correct companies can only be used with the appropriate application client.

s

The following is a description of the individual fields on the window:

*Profile ID*

This field contains a unique identifier for each Company Login Filter profile in the system. The lookup button can be clicked to select from existing profile IDs.




*Note that the Profile IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

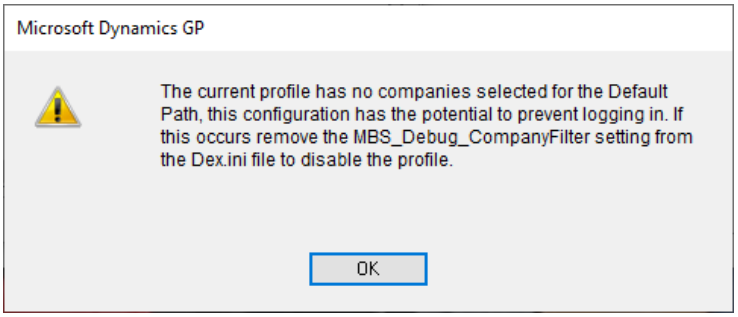
*Profile Name*

This field contains a description for the Company Login Filter profile.

*Enable current Profile on this workstation*

Selecting this field will set the MBS_Debug_CompanyFilter Dex.ini Setting for the current workstation to the current Profile ID.

If enabling a Profile ID without any companies selected for the Default Path, the following warning will be displayed:




*If you get locked out of Microsoft Dynamics GP because Company Login Filter removes access to all companies even when logging in as the 'sa' user, edit the Dex.ini file and remove the MBS_Debug_CompanyFilter setting.*

*Roll out Profile using Dex.ini Configuration*

Selecting this option will insert a record into the Dex.ini Configuration window to automatically roll down the Profile to all workstations and servers in the system.

*Share User Settings for all Launch File Paths*

Selecting this option will allow the same user settings selected using the Users Button to be used for all Launch File Paths. This means that enabling or disabling a user only needs to be done once rather than for each Launch File Path.

*Show Disabled Companies*

Selecting this field will display disabled companies at the bottom of the drop-down list instead of removing them entirely with the prefix entered into the next field.

*Using the Show Disabled Companies option provides a visual indication of the companies a user has security access to but cannot use from the current application instance.*

*Auto select if only one Company*

Selecting this field will automatically select the company if there is only one company available after the filter has been applied.

*Hide "Remember this Company" checkbox*

Selecting this checkbox will hide the "Remember this Company" checkbox from the Company Login window.

*Prefix for Disabled Companies*

Use this field to define a prefix to be used when showing disabled companies instead of removing them from the drop-down list.

*Display Company Database*

This field is used to select if the Company Database is shown on the drop-down list and whether it is shown as a prefix or suffix to the Company Name.

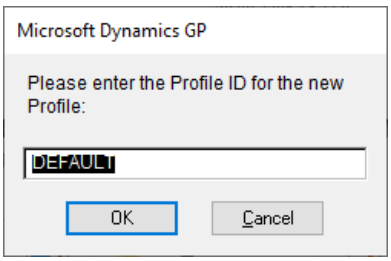
*Company Display Sort Order*

This field is used to select the order that the companies are listed in the drop-down list. This includes a Custom Defined Order, which can be set using the right hand Top, Up, Down and Bottom buttons.

The following is a description of the additional buttons on the window:

*Duplicate Button*

Use this button to duplicate the current profile ID to a new profile ID. This is useful when an existing profile ID is very similar to the new one you want to create.

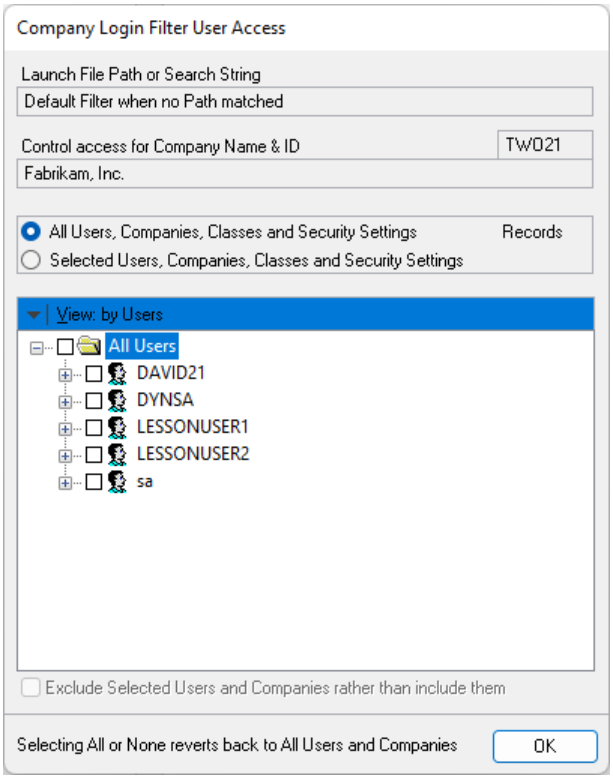



A new profile ID must be specified in the dialog which opens.

*Users Button*

Use this button to enable or disable access to a company based on Users and/or Companies, User Classes, Security Roles, Security Tasks or Security Modified Alternate IDs.

Depending on if the Share User Settings for all Launch File Paths option is selected, access can be control for all Launch File Paths or individually for each Launch File Path.



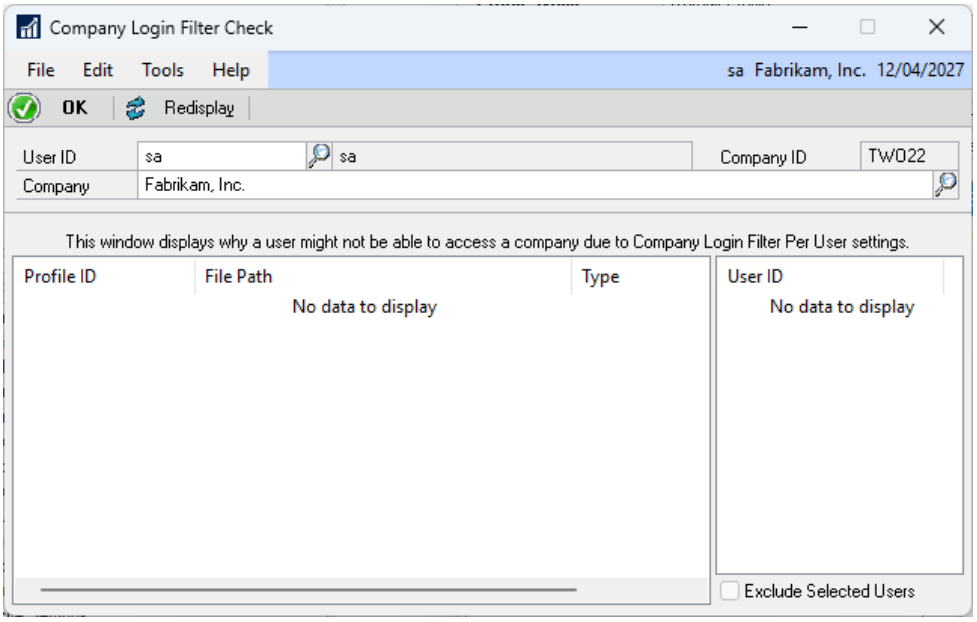

You can view this window by Users and/or Companies, User Classes, Security Roles, Security Tasks or Security Modified Alternate IDs and navigate the tree to select the options as required.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the disabling should not take effect.

### *Check User Button*

Use this button to open the Company Login Filter Check window which will display reasons that a user might not be able to access a company. This window is designed to help prevent Company Login Filter blocking access to the application.




*The Company Login Filter Check window will also open automatically if the current user is about to lock themselves out of the Microsoft Dynamics GP application due to settings in the Company Login Filter window. The Company Login Filter Check window can also be accessed from the User Access Setup window via the Additional menu.*

When setting up a Company Login Filter you can create a new profile ID for each application folder than is using a separate Dex.ini file. If you are using a single application folder with a single Dex.ini file with different Launch File names or paths, you can use a single profile ID with multiple paths specified.

When a profile is created, it will always have a default path created named "Default Filter when no Path matched". This default path will always be at the top of the list of paths.

You may add additional paths which are checked with a case insensitive "contains" comparison against the current Launch File path. The comparisons for the paths are executed in the order the paths are listed on the window. If no paths are matched, the default path will be used.

For each path in the left-hand pane, you can select which companies will be can be accessed in the right-hand pane.

You can add a Path using the Add Button or using the button drop-down list at the top of the left-hand pane.

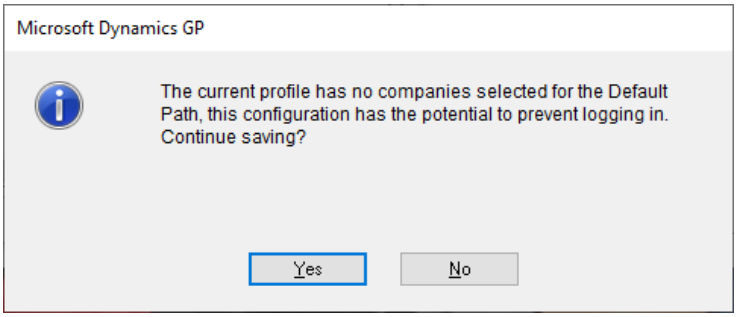
Company Login Filter allows the order of the paths to be changed using the left hand Top, Up, Down and Bottom buttons.

You can also edit an existing path with the Edit Button or double clicking on it. You can also remove a path with the Delete Button.

*Once the setup has been completed, upon login if the MBS_Debug_CompanyFilter Dex.ini Setting has a value and the profile can be found, then the paths are compared and a set of company access settings will be used to restrict access on the Company Login window. If there are no valid companies available, a message to say that will be displayed.*

*If you create a Company Login Filter configuration that denies you access to all companies and you can no longer log into Microsoft Dynamics GP, delete the MBS_Debug_CompanyFilter Dex.ini Setting to regain access and then adjust your settings as required.*

*To attempt to avoid this situation the following warning is displayed when saving a profile which does not have any companies enabled for the Default Path:*

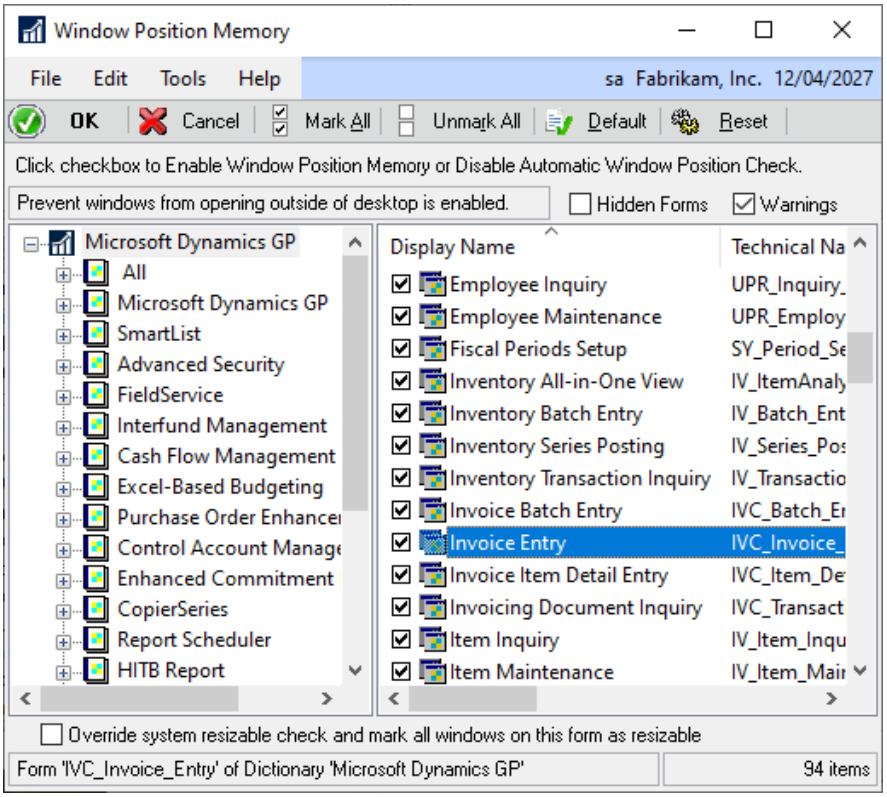


## Window Position Memory

You can open the Window Position Memory window by selecting Window Position Memory from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Window Position Memory from the Options button drop list on the main window. This is an Advanced Mode feature.

The Window Position Memory window is used to specify which windows in the Microsoft Dynamics GP application should remember their position, size and state on a per user basis. Any window (form) in any dictionary can be added to the list and all sub-windows on selected window (form) will be remembered.

You can also disable the Window Position Check functionality enabled in Administrator Settings (Prevent application windows from opening outside of the visible desktop area) on a per window basis for hidden windows that need to remain hidden.




Below is a description of the individual fields on the window:

*OK Button*

This button will save the settings and close the Window Position Memory window.

*Cancel Button*

This button will close the Window Position Memory window without saving any changes made.

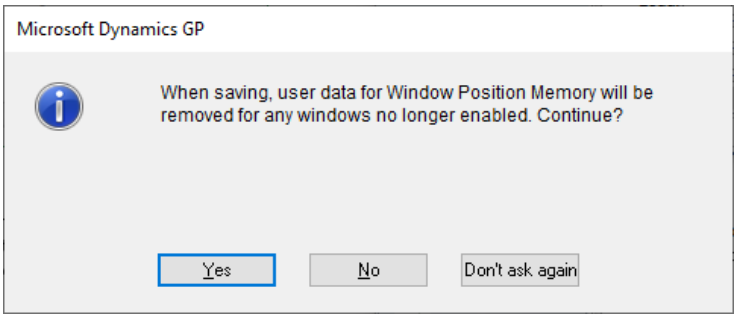
### *Mark All Button*

This button will select all windows (or all highlighted windows) currently showing in the right-hand pane.

### *Unmark All Button*

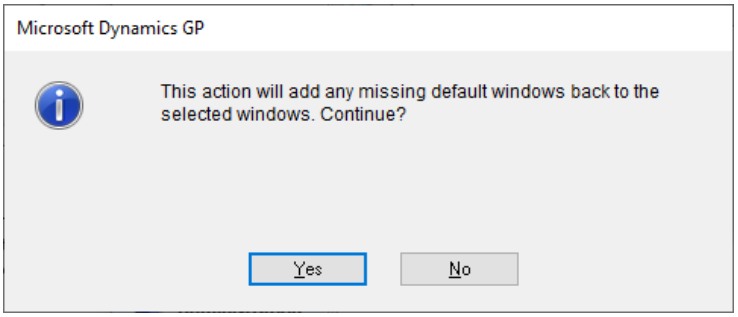
This button will de-select all windows (or all highlighted windows) currently showing in the right-hand pane.

If de-selecting a window that has user data associated with it, you will be warned that this data will be removed when the list of active windows is saved. Click "Don't ask again" if you wish the warning to be hidden for the rest of the time the window is open.



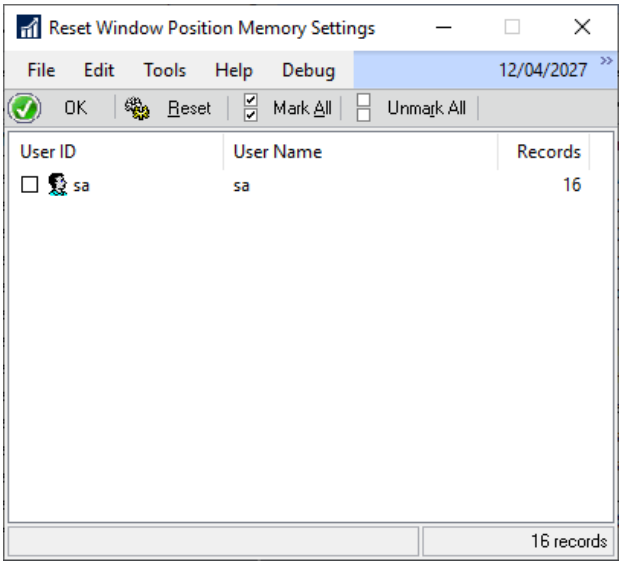

### *Default Button*

This button will add the default windows back to the list. The default windows include the main transaction, inquiry and cards windows from all core modules and the SmartList window.

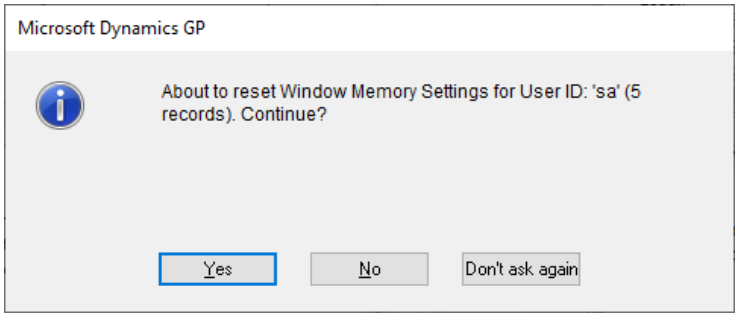


*Reset Button*

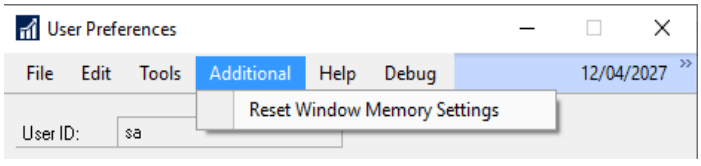
This button will open the Reset Window Position Memory Settings window.



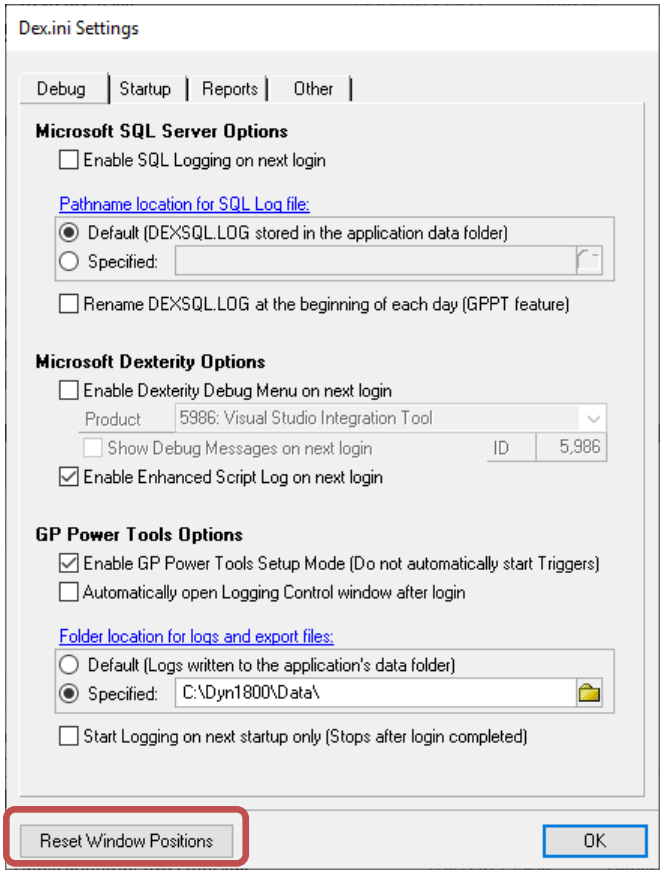

This window allows the administrator to reset the already stored window position, size and state data for the selected users. Once reset, the windows will open in the default position, size and will store settings again when it is closed.




Users can reset their own settings without needing the help of an administrator using the Reset Window Memory Settings option from the Additional menu on the User Preferences window.



Users can also reset their own settings from the Dex.ini Settings window using the Reset Window Positions button which resets both GP Power Tools windows and windows controlled by Window Position Memory.




### *Hidden Forms*

Use this checkbox to enable the display of windows normally hidden by the system. This might be required if the window you want to add cannot be found.

### *Warnings*

Use this checkbox to disable or re-enable the warnings when removing windows from Window Position Memory.

### *Override system resizable check*

Use this checkbox to override system resizable check and mark all windows on the selected form as resizable. Use this option when Window Position Memory fails to remember the new size of a window because it decided that the window was not resizable.

*When the Microsoft Dynamics GP node or All dictionary node on the left-hand tree is selected, the right-hand list will contain all of the currently active windows. When specific dictionary node is selected, the right-hand list will contain all of the currently active windows in that dictionary. When any other nodes are selected, the right-hand list will contain all windows in the application for the selected dictionary and series with the currently active windows showing as checked.*

*When Window Position Memory has been enabled for the SmartList window, GP Power Tools takes over control of handling the window from SmartList and makes it behave correctly. There is an issue on the latest versions of Microsoft Dynamics GP when the ribbon is enabled on the desktop client where the window size shrinks each time it is opened. There is also an issue when the SmartList window is closed while maximized. Both of these issues are fixed when GP Power Tools has control.*

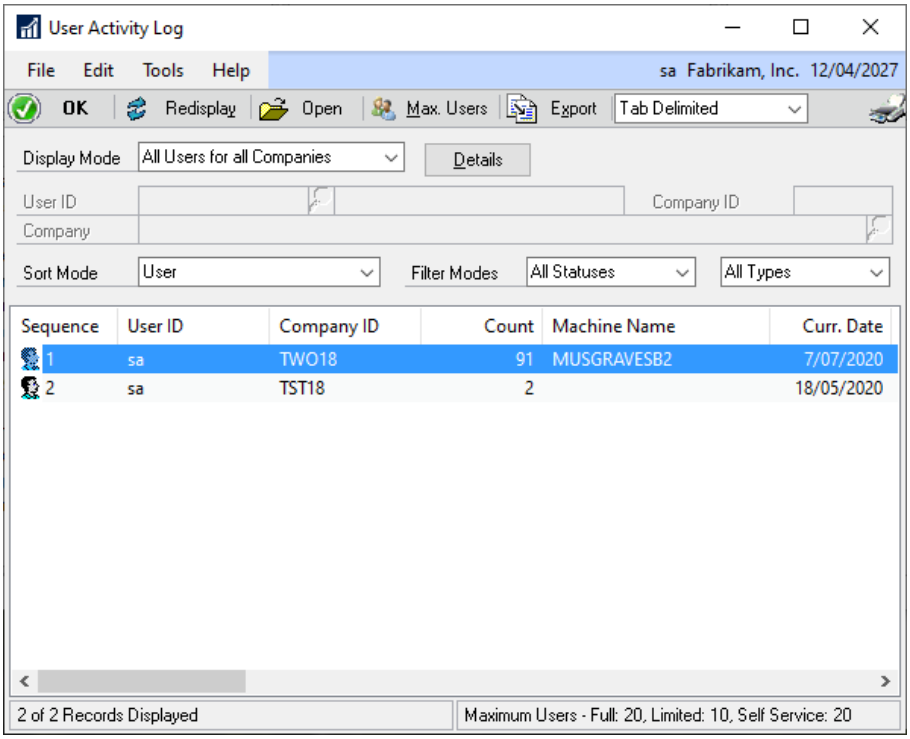
*The Window Position Memory feature can be disabled using the Usability Tab of the Administrator Settings window. Changing this option will take effect immediately on the current workstation and on next login for other workstations.*

*The Window Position Memory window can be used to disable specific windows from this feature if they are hidden windows which are now being displayed when they should remain hidden.*

## User Activity Log

You can open the User Activity Log window by selecting User Activity Log from the Setup section of the GP Power Tools Area Page or by selecting Administration >> User Activity Log from the Options button drop list on the main window. This is an Advanced Mode feature.

The User Activity Log window displays the data captured by the User Activity Tracking option which can be enabled from the Administrator Settings window using the Enable User Activity Tracking option.




Once the User Activity Tracking is enabled, all user login and logout events are tracked. The logging does not track individual events, but instead totals up the number of events so you can see which users are logging in the most. It also tracks the last three login events for a user. Data from the Automatic Logout feature are also tracked, Automatic Logout can be enabled from the Administrator Settings window.

Each event is tracked for the user and company, user, company and system wide, and you select how you want to view the data.

Below is a description of the individual fields on the window:

*Display Mode*

This drop-down list allows you to select whether your wish to view data for the selected user and company, for a specific user or company or for all users and companies.

The Machine Name Display Mode can be used if you want to see which workstations or servers are being used.

*User ID*
Use this field to select the User ID to display.

*Company*
Use this field to select the Company to display.

*Sort Mode*
This drop-down list can be used to select the order that the User Activity Log entries are initially displayed in. You can also adjust the sort after the data is displayed by clicking on the column headers.

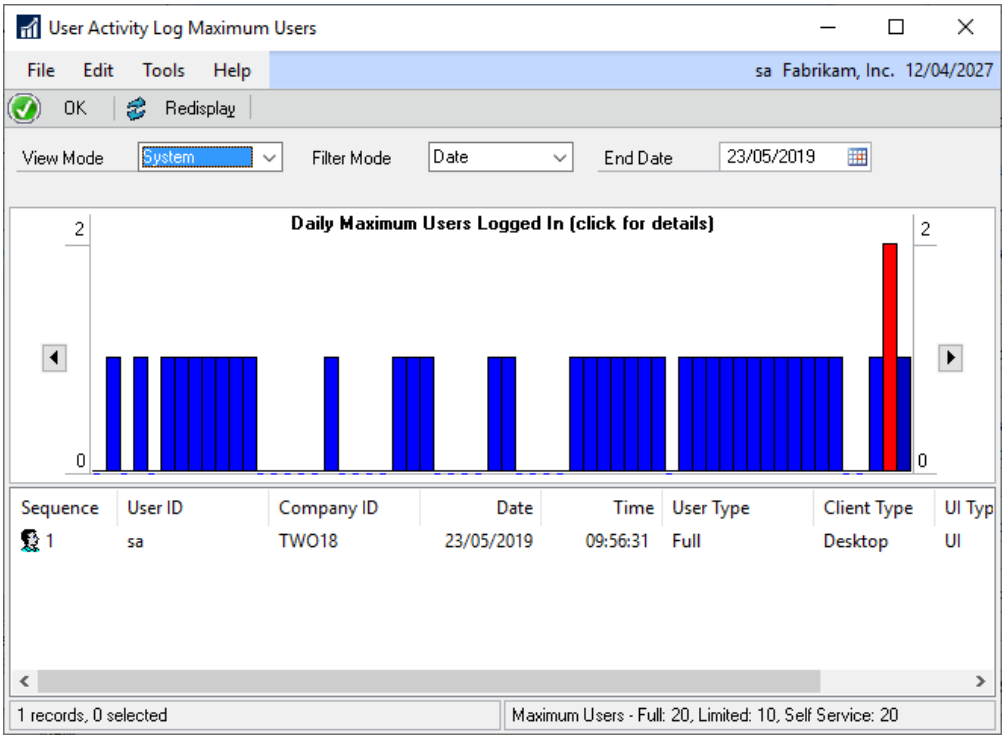
*Filter Modes*
These drop-down lists can be used to filter the User Activity Log entries. You can select to filter by User Status and User Type.

*OK Button*
This button will close the User Activity Log window.

*Redisplay Button*
This button can be used to redisplay the current contents of the User Activity Log data to the window.

*Open Button*
This button will open the User Setup window for the selected User.

*Max. Users Button*
This button will open the User Activity Log Maximum Users window.




This window displays a graph of the daily maximum sessions on a system, per user or per company basis. The graph can be viewed for a date range or showing the maximum values first.

The graph is auto scaling and hovering over any bar will show a tooltip with the date and maximum value on that date. Clicking on a bar will display a list of the logged in sessions when the maximum occurred.

### *Export Button*

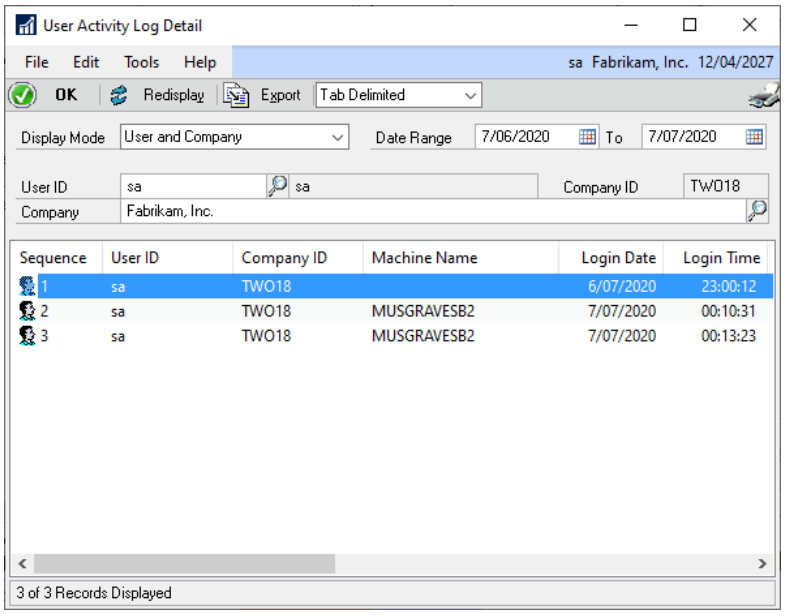
This button will allow the result set displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

### *Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

### *Details Button*

This button will open the User Activity Log Detail window to display individual records of each login and logout event. Turn the capture of this detailed data on from the Administrator Settings window.




The Automatic Logout feature records information in the User Activity Log. Below is a list of the columns shown in the log:

- Auto Cancel – Number of times Automatic Logout dialog cancelled.
- Auto Count – Number of times Automatic Logout attempted to exit.
- Auto Date – Date of last Automatic Logout attempt.
- Auto Time – Time of last Automatic Logout attempt.
- Auto Exit – Number of times Automatic Logout successfully exited.

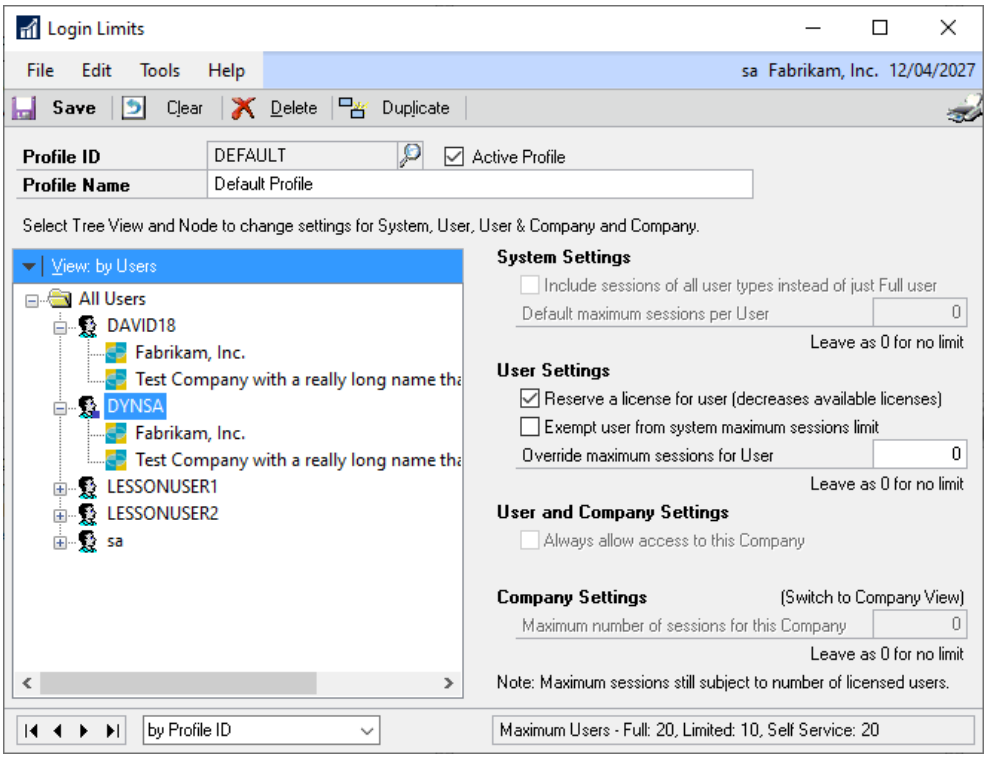
## Login Limits

You can open the Login Limits window by selecting Login Limits from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Login Limits from the Options button drop list on the main window. This is an Advanced Mode feature.

The Login Limits window is used to set up limits for how many sessions are allowed for a user logging into Microsoft Dynamics GP. While you can create more than one profile, only one profile can be marked as the Active Profile and be used at any one time. The login limits options can be set on a system wide, per user, per user and company and per company basis.

Examples of Use:

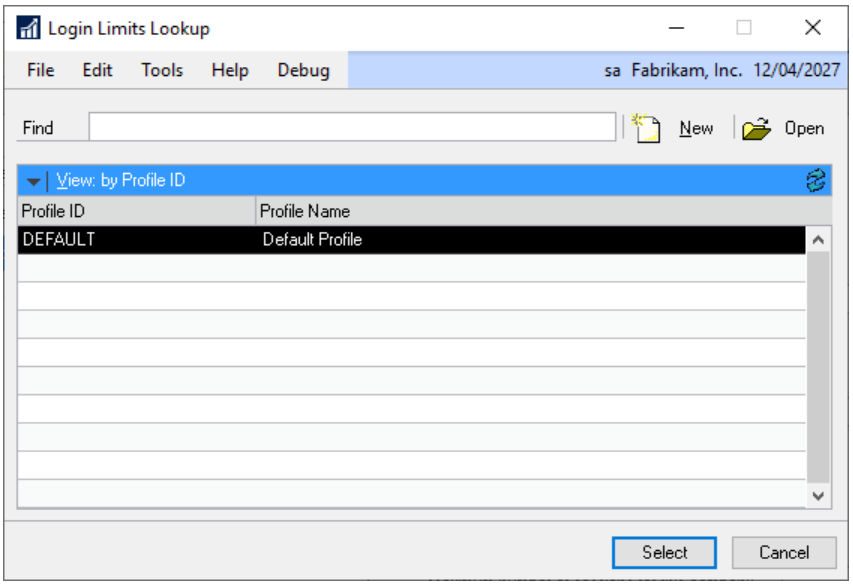
- You can limit users to one session system wide, but then mark the system administrator and company account accounts as exempt from the limits.
- You can reserve a license for a user to guarantee that they can always log into the system. This reduces to available licenses for other users.
- You can mark a company as always accessible for a user, thus allowing them to always access that company as well as one other company. (as per the one session limit specified above).
- You can also set a limit for the maximum number of sessions that can access specific companies, thus preventing too many sessions be used for one company meaning none are available for another company.



The following is a description of the individual fields on the window:

*Profile ID*

This field contains a unique identifier for each Login Limits profile in the system. The lookup button can be clicked to select from existing profile IDs.




*Note that the Profile IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*Profile Name*

This field contains a description for the Login Limits profile.

*Active Profile*

Selecting this checkbox marks this profile as active so it will be applied on next login.

The following options become enabled depending on the node selected in the tree in left-hand pane of the window. You can select a System Node, User Node, User and Company Node or Company Node.

*Include sessions for all user types instead of just Full user*

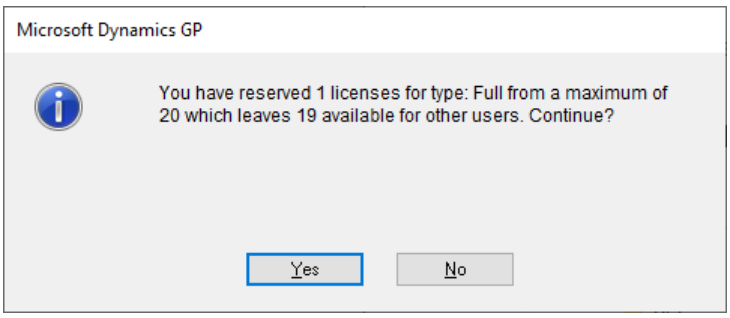
Selecting this field will adjust how the currently used sessions will be counted. Normally only Full Users are counted, but you can opt to include Limited or Service Users as well.

*Default maximum sessions per User*

Use this field to define the default system wide maximum number of allowed sessions per User. Leave as 0 for no limit.

*Reserve a license for user*

Marking this checkbox will reserve a license to guarantee that the selected user can always log into the system. When enabling this option, the following dialog shows how many reserved licenses you have configured and how many licenses are still available for other users.




*Exempt user from system maximum sessions limit*

Selecting this field will exempt the selected user.

*Override maximum sessions per User*

Use this field to define the override user level maximum number of sessions for the selected user. Leave as 0 for no limit.

*Always allow access to this Company*

Use this checkbox to allow the selected user to always have access to the selected company. This session is not included in the maximum session limits specified at the system or user levels.

*Maximum number of sessions for this Company*

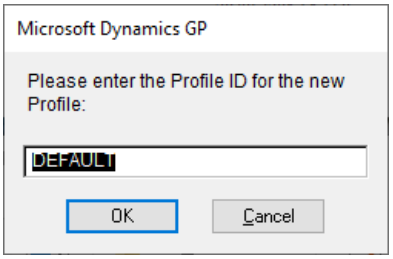
Use this field to define a maximum sessions limit for the selected company. Leave as 0 for no limit.

*To be able to set the Maximum number of sessions for this Company you will need to change the tree to the "by Company" view using the view button drop down list above the tree and then select the desired Company node.*

The following is a description of the additional buttons on the window:

*Duplicate Button*

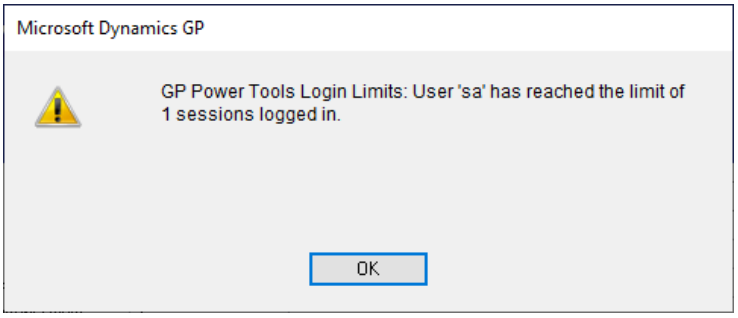
Use this button to duplicate the current profile ID to a new profile ID. This is useful when an existing profile ID is very similar to the new one you want to create.



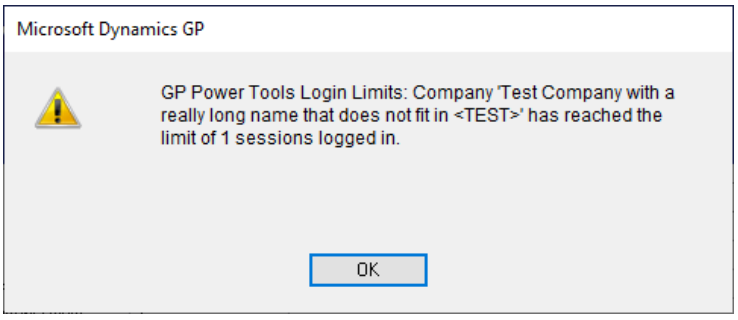

A new profile ID must be specified in the dialog which opens.

*Change the tree to the "by User (with settings)" or "by Company (with settings)" views using the view button drop down list above the tree to only see users and/or companies for which Login Limits has settings enabled. You can also identify which users and/or companies have settings by the colored markers on the bottom right corner of the icons on the tree.*

Once configured, the active profile will be checked at login and the user can be presented with a warning dialog when they click OK if they have exceeded the number of sessions they are allowed:




If the maximum number of sessions for a company has been reached, the following dialog will be displayed:

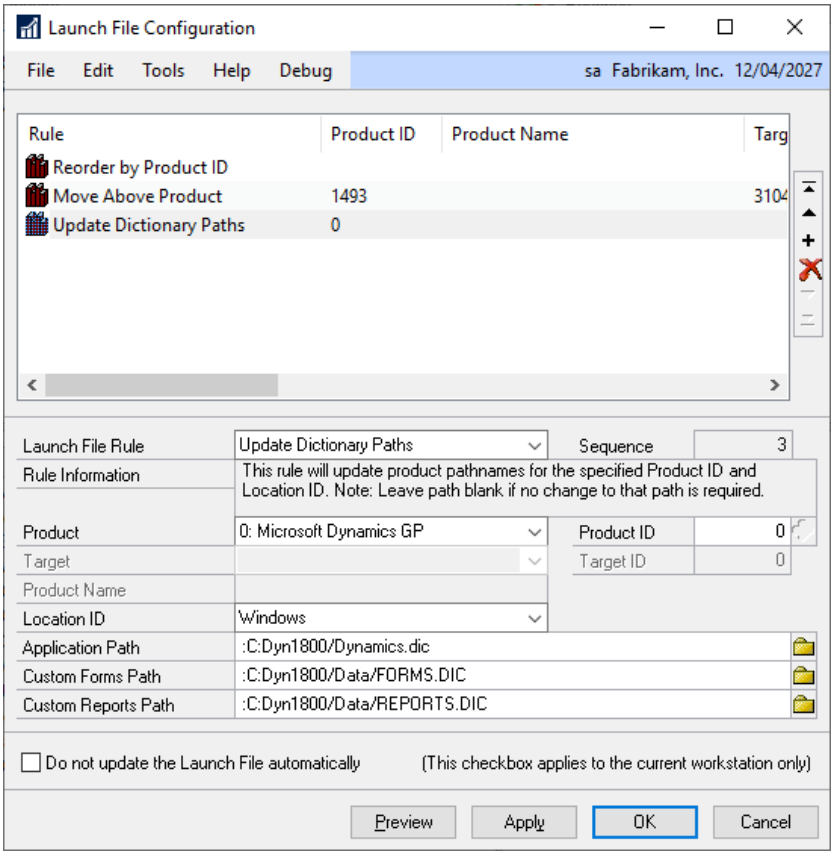



When the dialogs are displayed a log entry will also be written to the current GP Power Tools log file.

## Launch File Configuration

You can open the Launch File Configuration window by selecting Launch File Configuration from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Launch File Configuration from the Options button drop list on the main window. This is an Advanced Mode feature.

The Launch File Configuration window can be used to automate changes to Dynamics.set launch file for all workstations in the system. It works by defining rules for changes desired. These rules are checked on login and applied (if necessary) after creating a backup.




The following is a description of the individual fields on the window:

### *Rule List*

This list contains the rules to be checked on login. The columns show the details for each rule.

To edit a rule, just select the row in the list and make the desired changes. The rule will be automatically updated if all the rule field settings are valid.

To add a rule use the Add Button and select the rule to use from Launch File Rule drop down list. Then enter the desired changes. The rule will be automatically updated if all the rule field settings are valid.

Launch File Configuration allows the order of the rules to be changed using the Top, Up, Down and Bottom buttons. This controls the order in which the rules are applied.

You can also remove a rule with the Delete Button.

### *Launch File Rule*

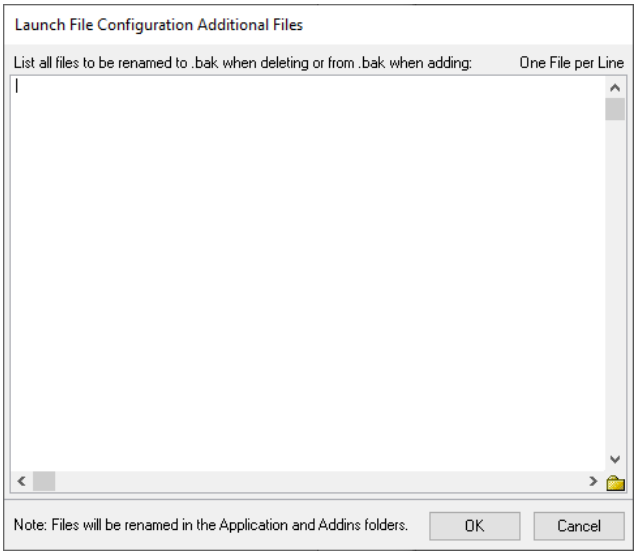
This drop down list is used to select the rule to apply to the launch file. Depending on the selection the Rule Information will be updated, and the Rule Fields will be enabled or disabled. Rules available are:

- Reorder by Product ID
- Reorder by Product Name
- Add New Product
- Remove Existing Product
- Rename Product Name
- Move Above Product
- Move Below Product
- Update Dictionary Paths
- Update Location ID Folders
- Update Dictionary Files
- Add Location ID
- Remove Location ID

### *Rule Fields*

These fields will be enabled and disabled depending on the Launch File Rule selected. The rule will be automatically updated if all the rule field settings are valid.

*When adding or removing a product, an expansion button will be available which opens the Launch File Configuration Additional Files window. Use this window to list additional files such as DLLs which need to be renamed with a .bak extension when removing a product or have the .bak extension removed when adding a product. Use the file path icon on the bottom right to select files from the file system.*

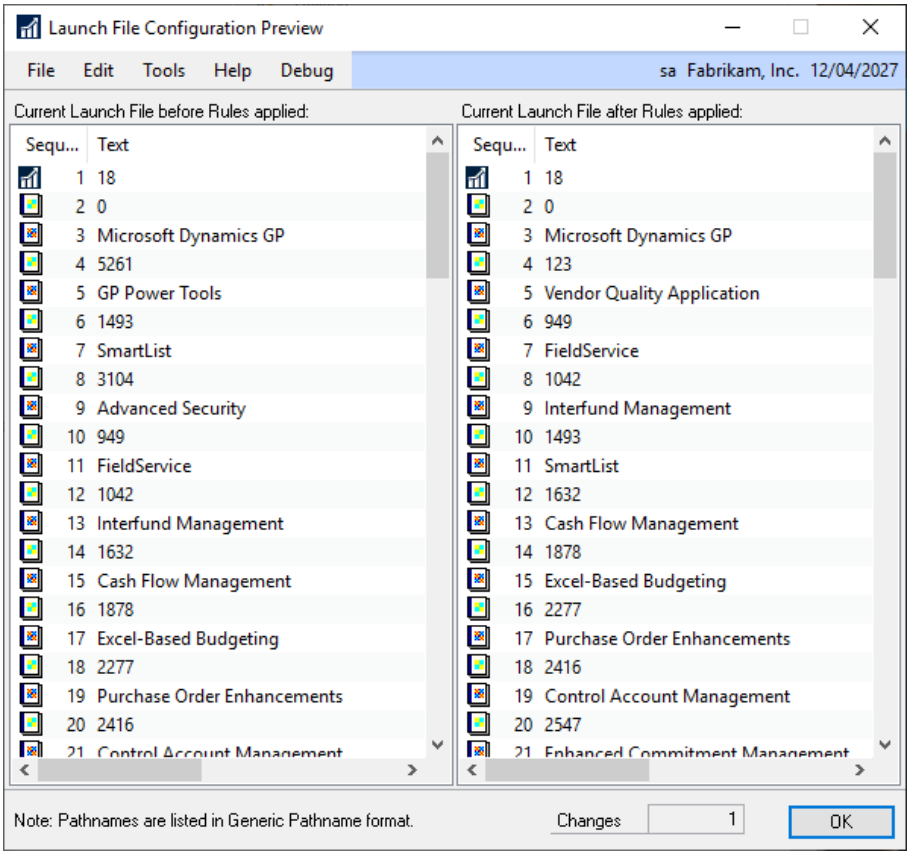


*Do not update the Launch File automatically*

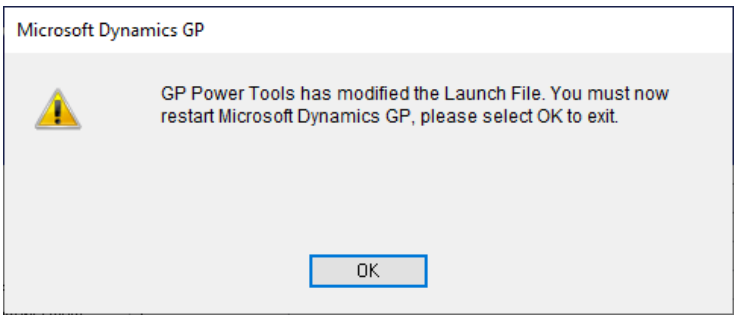
This field can be used on individual workstations to prevent GP Power Tools from automatically updating the Launch File. It will update the MBS_Debug_LaunchConfigurationOverride Dex.ini setting. This can be useful on test or administration workstations which might not want their Launch File changed.

The Apply Button can be used to save the changes to the setup files without closing the window.

Click the Preview button to view a preview of what changes would be made to the current workstation based on the rules defined. The Launch File Configuration Preview window will open. Changes to the rules are immediately reflected in the preview window.

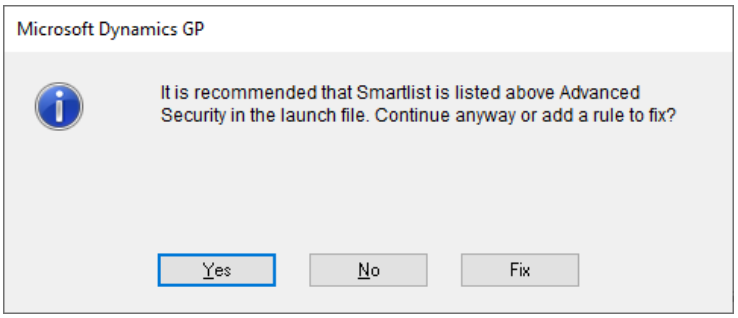



Launch File Rules specified in the Launch File Configuration window are checked against the current launch file (usually Dynamics.set) when a user logs in. If any changes are required, a backup of the existing launch file will be created, and a new updated launch file will be written. The user will then be notified and asked to restart Microsoft Dynamics GP.




*It is recommended to ensure that you do not have any rules which contradict each other as this could create an infinite loop where GP Power Tools keeps updating the Launch File each time Microsoft Dynamics GP is started and thus prevents access.*

*If you create rules which will leave the Advanced Security product above Smartlist in the Launch File, when saving you will be shown the dialog below which will offer to add a rule to fix this situation. If left unchanged, Smartlist can open a login dialog on startup before the actual Microsoft Dynamics GP login window opens.*

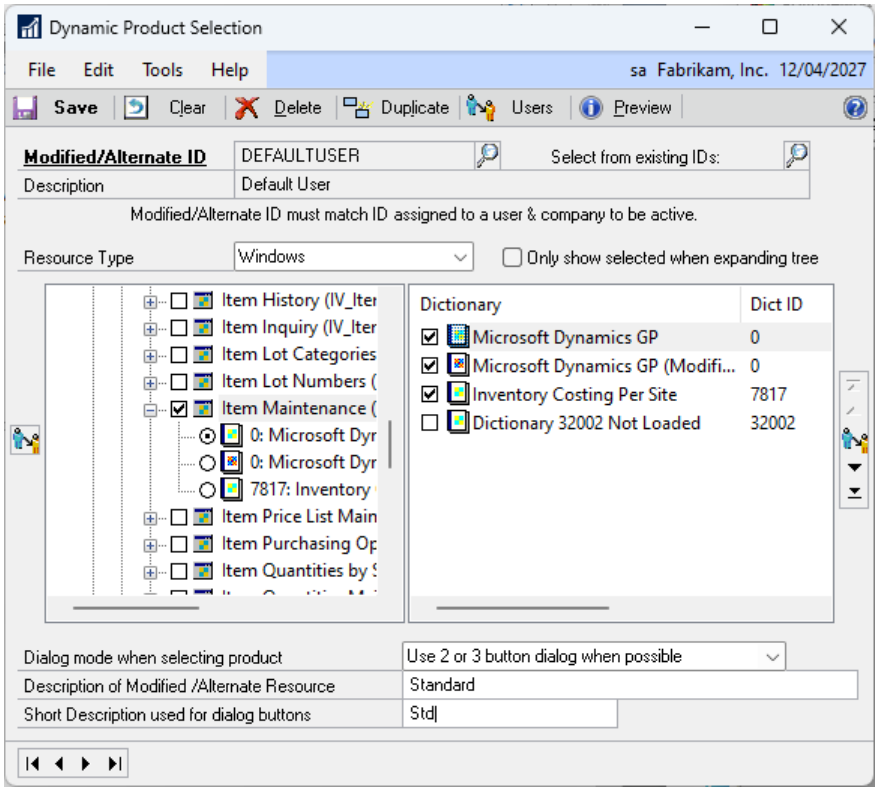


## Dynamic Product Selection

You can open the Dynamic Product Selection window by selecting Dynamic Product Selection from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Dynamic Product Selection from the Options button drop list on the main window. This is an Advanced Mode feature.

Dynamic Product Selection allows the dynamic selection of different versions of a window or report to be selected as the window or report is opened. This can be used to allow access to more than one version of a window (original, modified, alternate or modified alternate) without requiring security settings to be changed or logging in as another user.

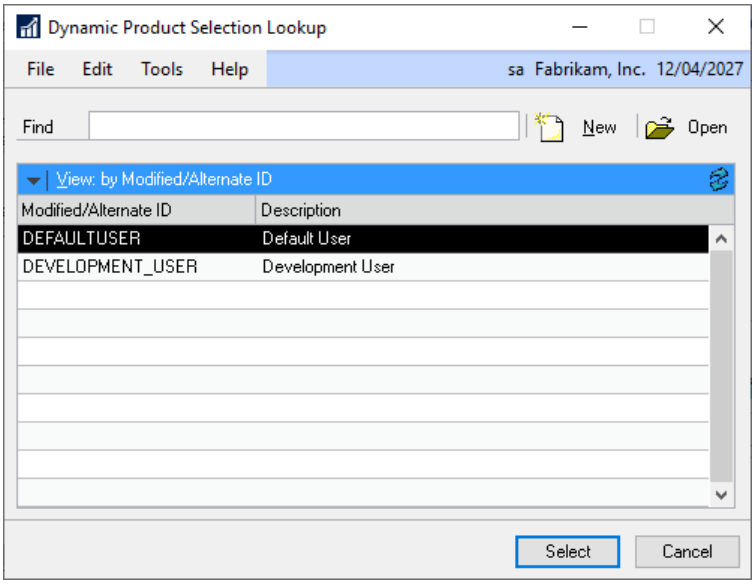
The Dynamic Product Selection window is used to configure which windows or reports have Dynamic Product Selection enabled and which versions of the window or report to offer for selection by the user. The settings are linked to the Microsoft Dynamics GP Alternate/Modified Forms and Report ID as used on the User Security window and can also have additional User and/or Company User Class, Security Role, or Security Task selection.



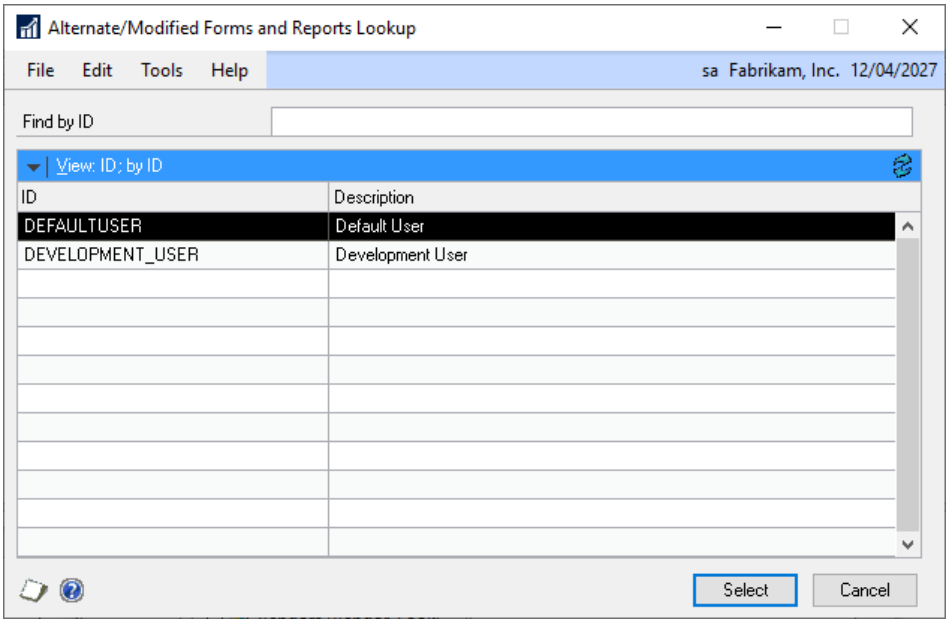

The following is a description of the individual fields on the window:

*Modified/Alternate ID*

This field contains a unique identifier for each Modified/Alternate ID in the system. The lookup button can be clicked to select from existing Modified/Alternate IDs.




While it is possible to have as many Modified/Alternate IDs as you wish, only those which match an existing Microsoft Dynamics GP Alternate/Modified Forms and Report ID will be used. To select from the list of existing Microsoft Dynamics GP Alternate/Modified Forms and Report IDs, use the lookup button on the right-hand side of the window.




*Note that the Modified/Alternate IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

### *Description*

This field contains the description from the matched Microsoft Dynamics GP Alternate/Modified Forms and Report ID, if it exists.

*Resource Type*

This drop down list selects whether to setup dynamic product selection for windows or reports resources.

*Only show selected when expanding tree*

This checkbox can be used to limit the Resource Tree to only show windows and reports which have been enabled for Dynamic Product Selection. This makes it simpler to locate active resources.

*Resource Tree*

Use the left-hand tree pane to select which resource should have Dynamic Product Selection enabled. The Users Button can be used to fine tune the Users and/or Companies User Class, Security Role, or Security Task the resource is enabled for beyond that they are using the specified Modified/Alternate ID.

*Selection List*

Use the right-hand list pane to select which versions of the resource are to be made available by Dynamic Product Selection.

The order of the choices to be changed using the left hand Top, Up, Down and Bottom buttons.

You can also fine tune the selections based on User and/or Company, User Class, Security Role, or Security Task using the User Button. Once clicked the Enabled for Users window will open, see the section below for more details.

*Dialog mode when selecting product*

Controls whether keyboard entry dialog should be used even when there are only two or three choices which can use a button dialog. Using the keyboard entry dialog uses one dialog to display a list of available options and a second dialog to enter the selection desired.

*Description of Modified/Alternate Resource*

Use this field to enter a description to display to the user rather than the name of the dictionary.

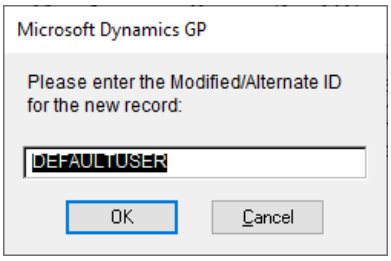
*Short Description used for dialog buttons*

Use this option to change the labels used on the dialog buttons rather than a simple letter of the alphabet. On the keyboard entry dialog, you can use the number of the selection or type in the short description.

The following is a description of the additional buttons on the window:

### *Duplicate Button*

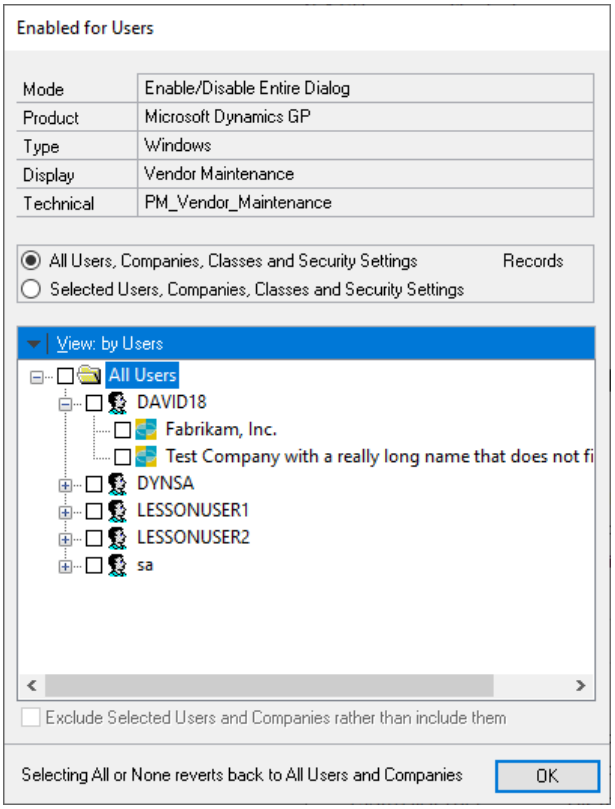
Use this button to duplicate the current Modified/Alternate ID to a new Modified/Alternate ID. This is useful when an existing Modified/Alternate ID is similar to the new one you want to create.




A new Modified/Alternate ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies should have the current resource enabled. Once clicked the Enabled for Users window will open.



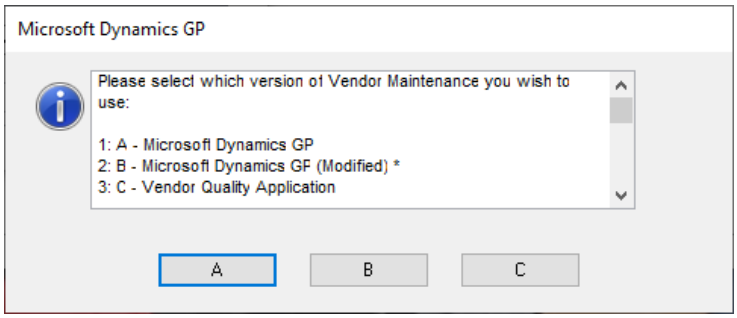

You can view this window by users, by companies or by user classes and navigate the tree to select the user and company combinations as required.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies.*

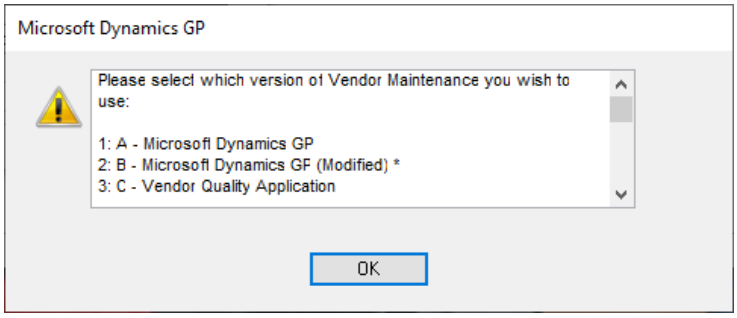
The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom dynamic product selection should not be enabled.

Once configured, when a user opens a window or report the Alternate/Modified Forms and Reports ID being used for the current user and company matches a Modified/Alternate ID and the resource has Dynamic Product Selection enabled and has more than one selection available, a dialog will be displayed and the user can make a selection of which version of the window or report they wish to open.

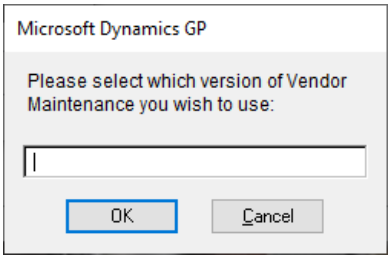
If there are three or less options and the button dialog mode is selected, then a button dialog will be displayed.




If there are more than three options or the keyboard dialog mode is selected, then two dialogs will be used, the first dialog with a list of options will be displayed:



Then a second dialog is displayed to allow the keyboard entry of the desired choice.



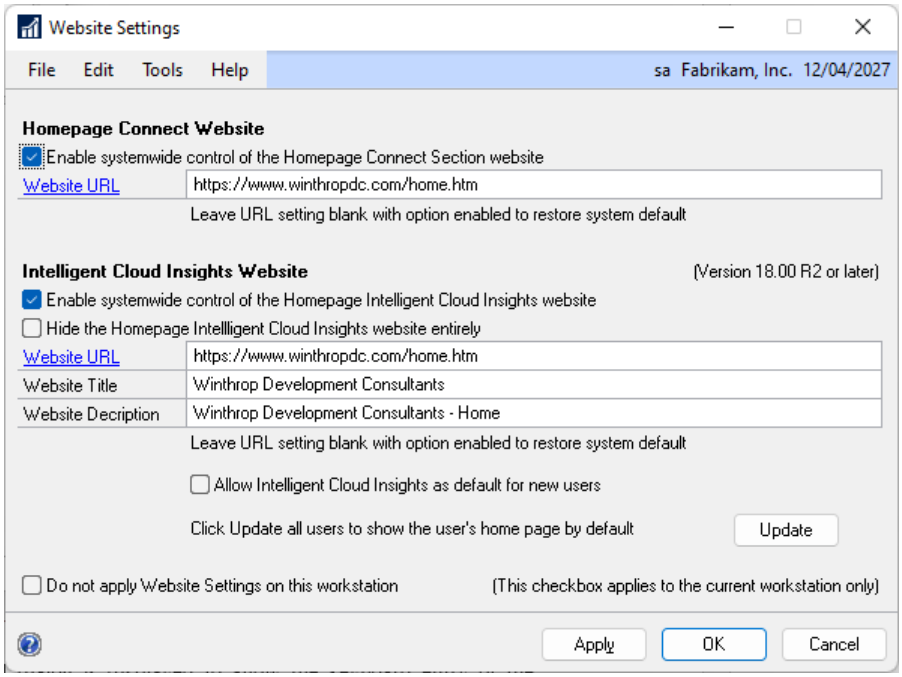

*Note a valid selection must be entered to proceed past this dialog, or you can select cancel to open the default version as controlled by security settings (displayed with the asterisk in the first dialog).*

Dynamic Product Selection handles checks by third party products of the security tables to ensure they correctly identify which version of a window is currently open. This prevents triggers running on the incorrect version of a window and generating errors.

## Website Settings

You can open the Website Settings window by selecting Website Settings from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Website Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

Website Settings allows overriding of the default webpage settings for the Connect and Intelligent Cloud Insights (GP 2018 R2 or later) homepage sections. These settings are system wide and achieved without using Dex.ini settings or Modifier.




The following is a description of the individual fields on the window:

*Enable systemwide control of the Homepage Connect Section website*
Mark this checkbox to enable changing of the website URL for the Connect Homepage section.

*Connect Section Website URL*
Enter the Website URL to use for the Connect Homepage section.

*Enable systemwide control of the Homepage Intelligent Cloud Insights Section website*
Mark this checkbox to enable changing of the website URL for the Intelligent Cloud Insights Homepage section.

*Hide the Homepage Intelligent Cloud Insights website entirely*
Mark this checkbox to hide Intelligent Cloud Insights Homepage section entirely.

*Intelligent Cloud Insights Section Website URL*

Enter the Website URL to use for the Intelligent Cloud Insights Homepage section.

*Intelligent Cloud Insights Section Website Title*

Enter the Website Title to use for the Intelligent Cloud Insights Homepage section.

*Intelligent Cloud Insights Section Website Description*

Enter the Website Description to use for the Intelligent Cloud Insights Homepage section.

*Allow Intelligent Cloud Insights as default for new users*

By default, Microsoft Dynamics GP makes Intelligent Cloud Insights the default view on the homepage for a newly created user. GP Power Tools changes the default back to the user's homepage. Use this checkbox to restore the Microsoft Dynamics GP default behavior.

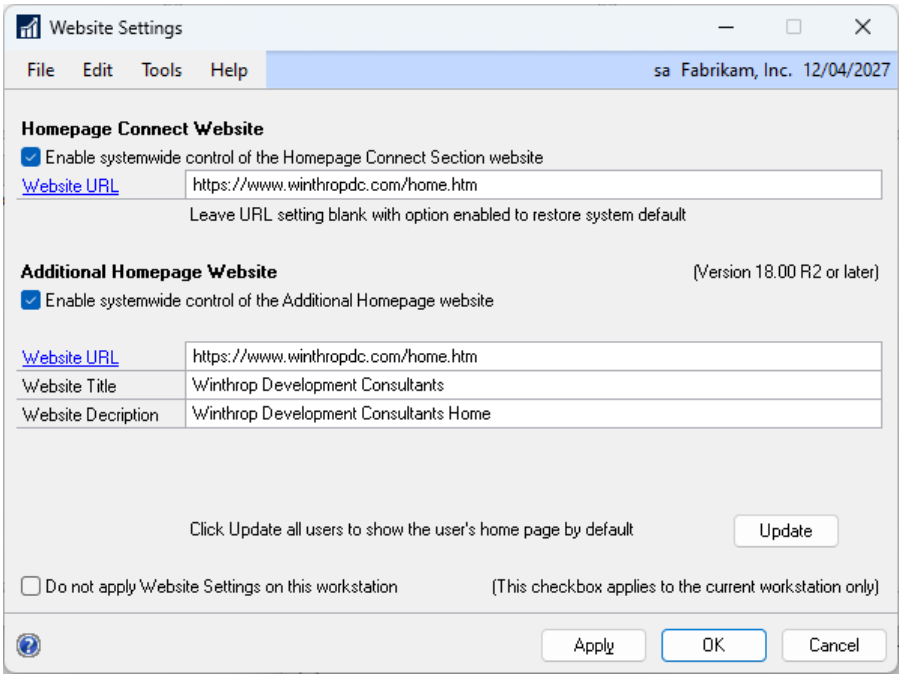
*Update Button*

Use this button to set all users to show the user's homepage rather than Intelligent Cloud Insights when Microsoft Dynamics GP loads.

*Do not apply Website Settings on this workstation*

Enabling this checkbox on a workstation uses the MBS_Debug_DisableWebsiteSettings Dex.ini setting to disable any website setting changes for the workstation.

*For Microsoft Dynamics GP v18.4 or later which removes Intelligent Cloud Insights, GP Power Tools will allow adding back of a custom website url instead of repurposing Intelligent Cloud Insights.*

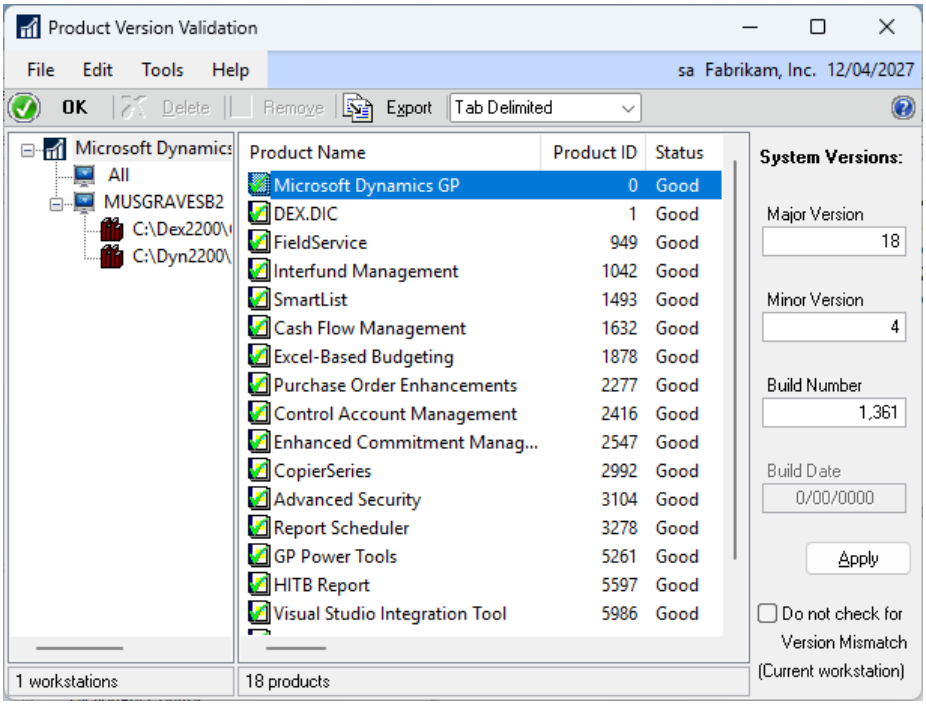


## Product Version Validation

You can open the Product Version Validation window by selecting Product Version Validation from the Setup section of the GP Power Tools Area Page or by selecting Administration >> Product Version Validation from the Options button drop list on the main window. This is an Advanced Mode feature.

Product Version Validation serves multiple purposes in the system. It tracks the product dictionary versions of all products installed on every workstation or server instance. It uses this information to ensure that the resource data cached by the Resource Explorer is automatically kept up to date when a dictionary is installed or updated.

Product Version Validation also keeps track of the latest product dictionary versions installed on a system and will provide a mismatch warning dialog when a workstation instance logs in with a different version or build number. Finally, for some GP Power Tools functions which require all product dictionaries installed to function correctly, it can check if any dictionaries are not installed.




The following is a description of the individual fields on the window:

*System Versions*

These fields can be used to update the version and build details stored for a product at the system level. Use this if you need to restore the details back to previous values. Changes must be applied before changing product.

*Do not check for Version Mismatch*

Use this checkbox to stop Product Version Validation display the warning dialog when a mismatch is found on this workstation. It uses the MBS_Debug_ProductVersionOverride Dex.ini setting.

*Apply Button*

Changes to the System Versions fields must be applied using this button to be saved.

*Delete Button*

Use this button to remove records for a product no longer installed on a workstation or remove a product from the system level. If the product is installed on a workstation instance it will be added back to that workstation instance and to the system level on next login.

*Remove Button*

Use this button to remove all records for the selected workstation or workstation instance. If the workstation logs in again, the records for that workstation will be added back.

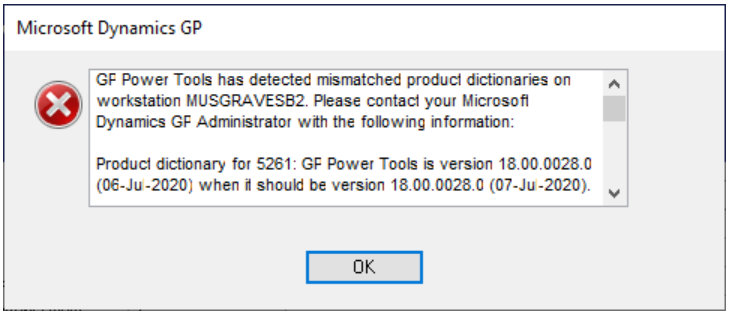
*Export Button*

This button will allow the result set displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

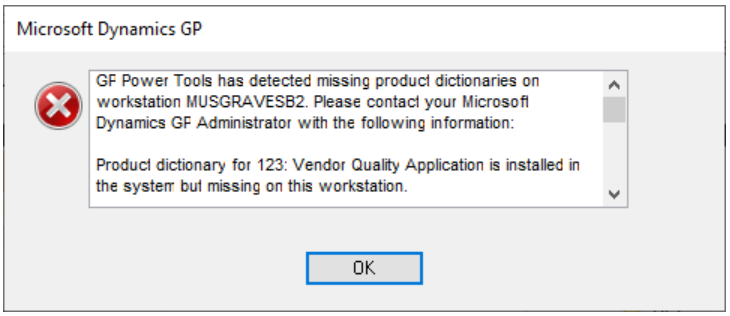
*Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

*If a workstation logs in with a different product dictionary version to the data stored at the system level, the following mismatch warning dialog will be displayed. This dialog can be resolved by installing the correct version of the product on the workstation, or by editing the System Versions details if they are incorrect.*



*If attempting to access a window in GP Power Tools which requires all product dictionaries installed and a product is not installed, the following missing warning dialog will be displayed. This dialog can be resolved by installing all the products on the workstation, or by deleting the product from the system level if they are no longer being used.*



## Additional Administrator Features

GP Power Tools adds some extra features to help administrators. Below is a summary of the features:

### Security Resource Descriptions

When opening GP Power Tools security related windows, the Security Resource Descriptions (SY09400) syCurrentResources table is updated to include resources from any missing or updated dictionaries and for resources types not updated by core code.

### SUPERUSER Security Task and Role

When GP Power Tools updates the Security Resource Descriptions table, it also creates and maintains a SUPERUSER Security Role and SUPERUSER Security Task. The SUPERUSER Security Role is designed to be used instead of the POWERUSER Security Role. Its advantage is that is uses the security system but grants access to everything instead of bypassing the security system like POWERUSER.

### SUPERUSER Workflow Setup

The Workflow Setup looks for POWERUSER for some features, GP Power Tools allows the same features to be used by a SUPERUSER. This allows Workflow Setup to be used without needing to go back to a POWERUSER.

### User Company Access Fix

GP Power Tools fixes the issue which can cause an error when changing User Company access when the settings in SQL Server and Microsoft Dynamics GP do not match

### User Setup Additional Information

GP Power Tools adds a window to the User Setup window to allow for the storage of additional data against a user. For more information see the setup options in the Administrator Settings window.

# Chapter 5: Developer Tools Features

This chapter includes the following sections:

- *Runtime Executer*
- *SQL Executer*
- *.Net Executer*
- *Project Setup**
- *Automatic Trigger Mode*
- *Trigger Setup**
- *Runtime Execute Setup**
- *SQL Execute Setup**
- *.Net Execute Setup**
- *Snippet Setup**
- *Parameter Lists**
- *Messages Setup**
- *Dynamic Trigger Logging**
- *Virtual Fields**
- *Additional Developer Features*

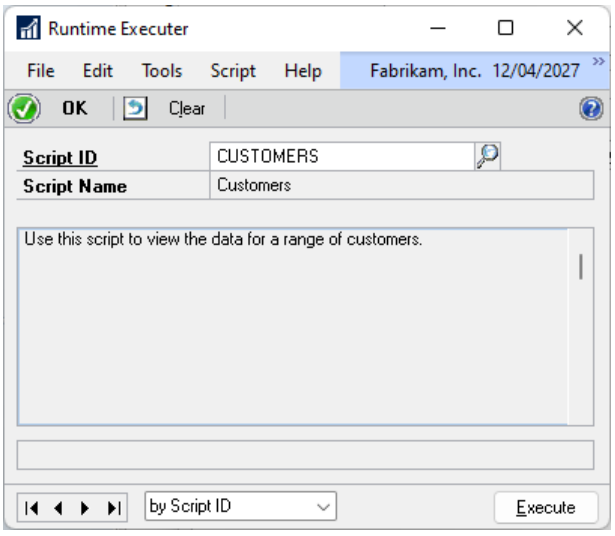
** Advanced Mode Feature*

## Runtime Executer

You can open the Runtime Executer window by selecting Runtime Executer from the Inquiry section of the GP Power Tools Area Page or by selecting Scripting >> Runtime Executer from the Options button drop list on the main window.

The Runtime Executer window can be used to run any Dexterity sanScript script created with the Runtime Execute Setup window which has been marked as Published to Executer Window.

*Scripts can only be executed from this window and cannot be viewed, edited or deleted. This window is designed to be used to expose specific scripts to be used by standard users, without needing to give them access to the Runtime Execute Setup window. The Long Description is displayed on the window.*




The following is a description of the individual fields on the window:

*Script ID*

This field contains a unique identifier for each Runtime Execute Setup script in the system. The lookup button can be clicked to select from existing published script IDs.

*Execute Button*

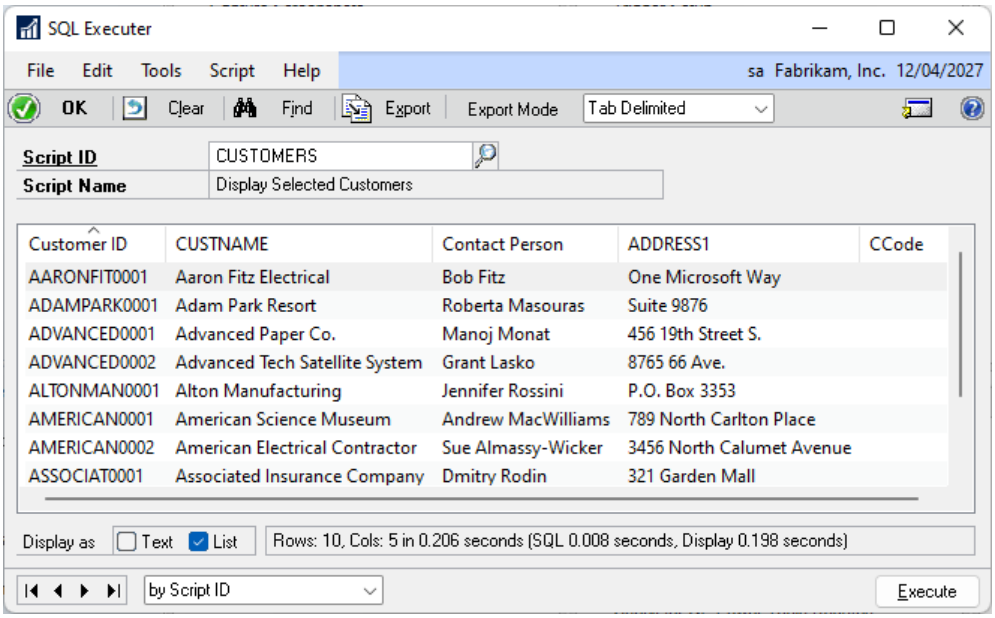
Use this button to execute the script.

## SQL Executer

You can open the SQL Executer window by selecting SQL Executer from the Inquiry section of the GP Power Tools Area Page or by selecting Scripting >> SQL Executer from the Options button drop list on the main window.

The SQL Executer window can be used to run any Transact SQL statements created with the SQL Execute Setup window which has been marked as Published to Executer Window.

*Scripts can only be executed from this window and cannot be viewed, edited or deleted. This window is designed to be used to expose specific scripts to be used by standard users, without needing to give them access to the SQL Execute Setup window. The Long Description is displayed on the window.*




The following is a description of the individual fields on the window:

*Script ID*

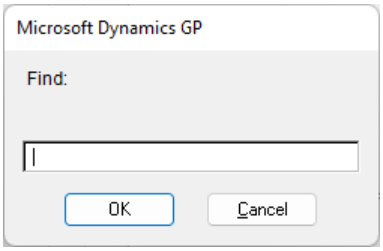
This field contains a unique identifier for each SQL Execute Setup script in the system. The lookup button can be clicked to select from existing published script IDs.

*Execute Button*

Use this button to execute the script.

*Find Button*

Use this button to search the results list for the specified search string. You can select to search All Columns or the current Sort Column and whether the search should be a Contains search or a Begins With search.




*Export Button*

This button will allow the result set displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

*Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

*Gotos Button*

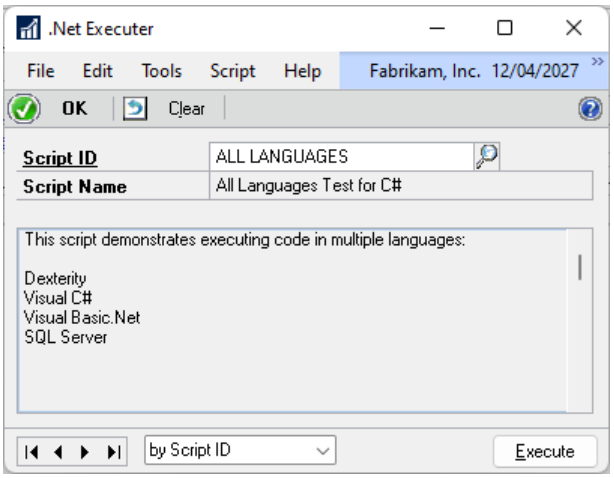
Use this button drop-down menu to execute a SQL Goto on the selected rows in the returned data. You can also right click on the results list.

## .Net Executer

You can open the .Net Executer window by selecting .Net Executer from the Inquiry section of the GP Power Tools Area Page or by selecting Scripting >> .Net Executer from the Options button drop list on the main window.

The .Net Executer window can be used to run any Visual C# or Visual Basic.Net script created with the .Net Execute Setup window which has been marked as Published to Executer Window.

*Scripts can only be executed from this window and cannot be viewed, edited or deleted. This window is designed to be used to expose specific scripts to be used by standard users, without needing to give them access to the .Net Execute Setup window. The Long Description is displayed on the window.*




*To be able to execute .Net scripts, the WinthropDC.GpPowerToolsVC.dll and WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

The following is a description of the individual fields on the window:

*Script ID*

This field contains a unique identifier for each .Net Execute Setup script in the system. The lookup button can be clicked to select from existing published script IDs.

*Execute Button*

Use this button to execute the script.

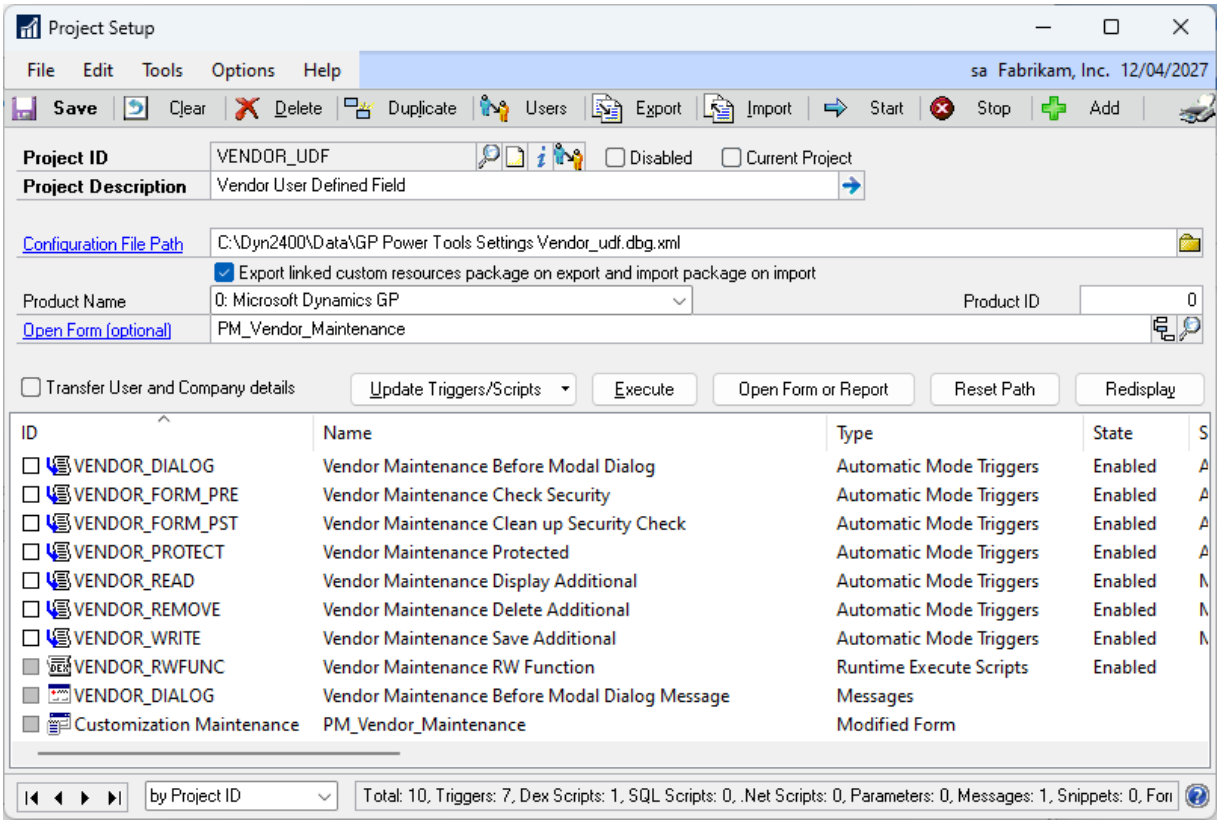
## Project Setup

You can open the Project Setup window by selecting Project Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Project Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The Project Setup window can be used to group together multiple triggers, scripts and parameter lists into a single development or customization project which can be easily exported without needing to manually select the resources on the Configuration Export/Import window.

The supported resources are listed below.

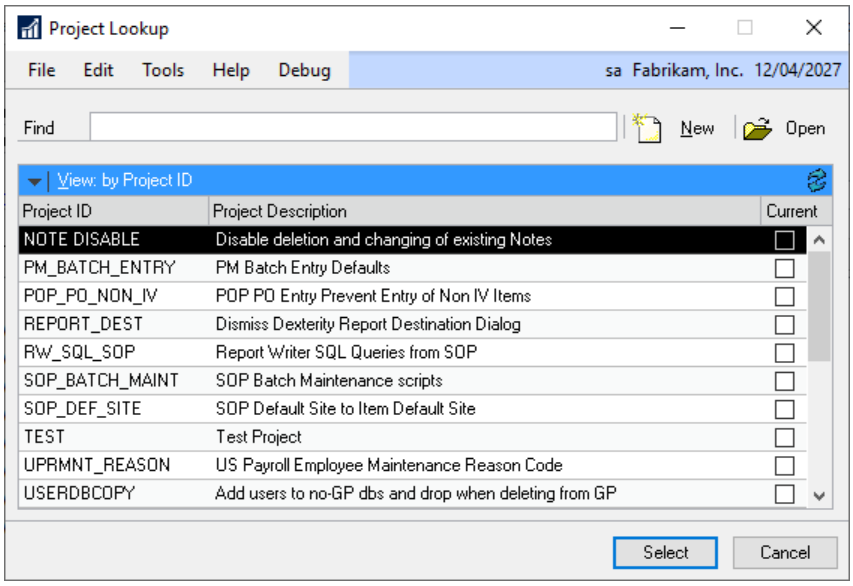
- Trigger Setup triggers
- Runtime Execute Setup scripts
- SQL Execute Setup scripts
- .Net Execute Setup scripts
- Parameter Lists
- Messages
- Snippets
- Form Controls
- Passwords
- Customization Maintenance resources



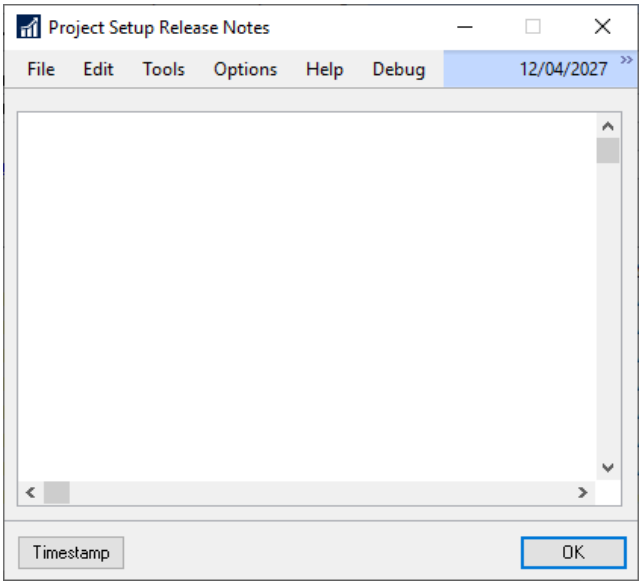
The following is a description of the individual fields on the window:

*Project ID*

This field contains a unique identifier for each Project Setup project in the system. The lookup button can be clicked to select from existing project IDs.



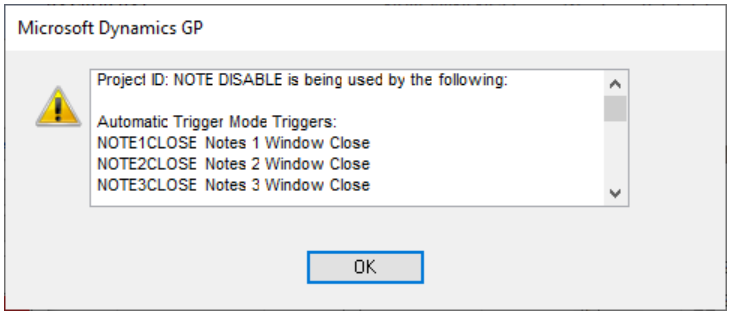

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Project IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*Project Information*

Click this button to display what resources are using this project.

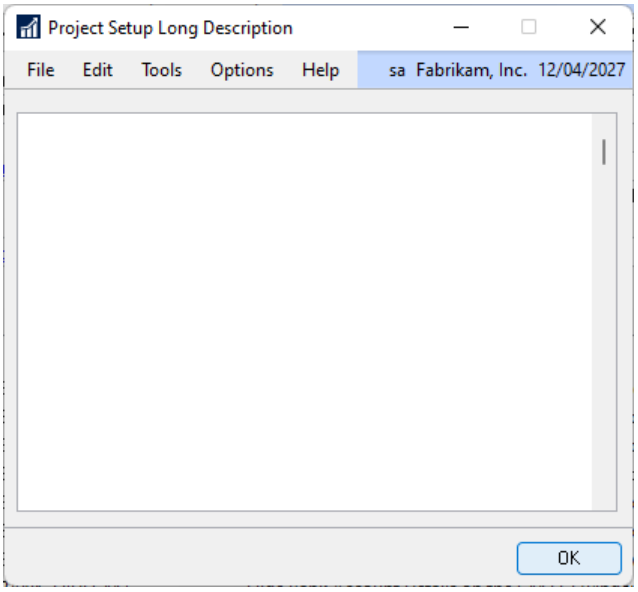



*Project Description*

This field contains a description of the project.

*Long Description*

Use the Project Description expansion button to open the Long Description window. Use this field for a more detailed description of the project.




*Disabled*

This checkbox disables the project by preventing triggers in the project from starting automatically on startup. It can be used rather than disabling each trigger individually.

*Current Project*

This checkbox indicates that this project is the current project and will be automatically loaded into the window when it is first opened.

*Configuration File Path*

This is the file name used for exporting. The file should use the extension .dbg.xml. The path is automatically generated based on the Project ID, but can be manually changed, if desired.

*Export Linked Custom Resources*

Use this checkbox to enable the importing and exporting of Customization Maintenance package files when importing and export the project.

*Product Name*

Use the Product Name, Product ID and Open Form fields to add a default form for the project. This form can be opened quickly by using the Open Form prompt drill down. Its only function is to provide quick and convenient navigation to a form for testing purposes.

*Product ID*

Use the Product Name, Product ID and Open Form fields to add a default form for the project. This form can be opened quickly by using the Open Form prompt drill down. Its only function is to provide quick and convenient navigation to a form for testing purposes.

*Open Form*

Use the Product Name, Product ID and Open Form fields to add a default form for the project. This form can be opened quickly by using the Open Form prompt drill down. Its only function is to provide quick and convenient navigation to a form for testing purposes.

*Transfer User and Company details*

This checkbox selects whether the user and company selection for triggers is exported when the trigger is exported.

*Project Component List*

This list shows all the components that make up the project. Double click on any item to open that item in the appropriate window. Use the checkbox on triggers to manually start or stop triggers.

*Right click to perform actions against the selected resources (use control and shift keys to multi-select). You can Open, Duplicate, Rename or Delete a resource as well as change settings without opening the resource, such as Disabling, Automatic Start Status and Minimize vs Full logs*

The following is a description of the additional buttons on the window:

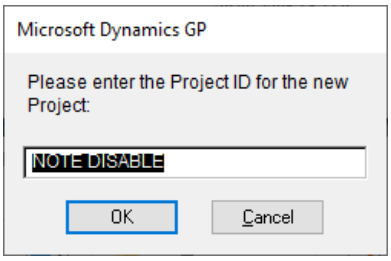
*Delete Button*

Use this button to delete the current project ID. You can select to delete just the project header or the entire project along with its components (except Customization Maintenance resources).

*A project that is currently linked to triggers, scripts or parameter lists cannot be deleted. If you attempt to delete a project while it is still in use, the information dialog shown above will also open to show you where the project is used.*

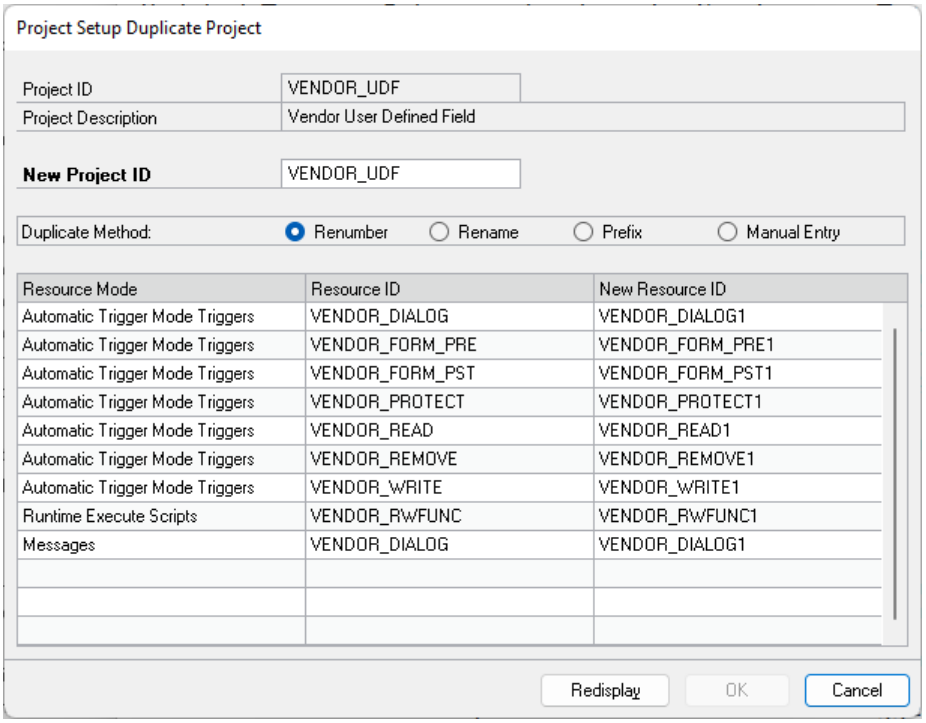
### *Duplicate Button*

Use this button to duplicate or rename the current project ID. You can select to duplicate the project header or the entire project or rename the project. This is useful when an existing project ID is very similar to the new one you want to create, or you wish to rename a project without having to update each of its components.




A new project ID must be specified in the dialog which opens.

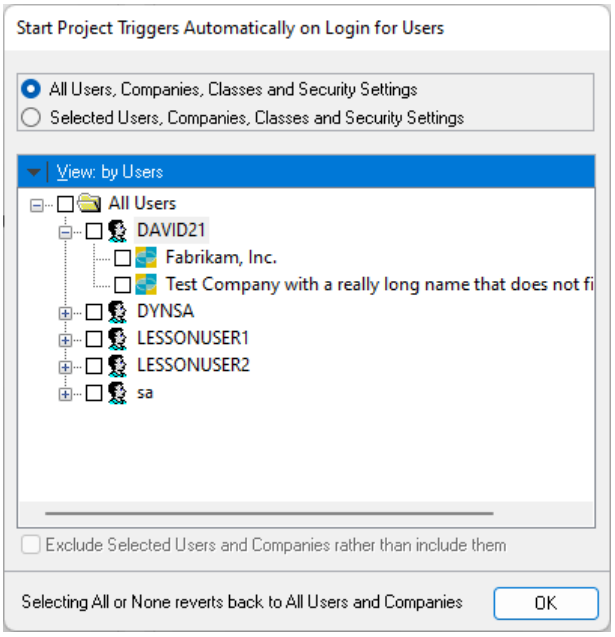
If you select to duplicate the entire project the following Project Setup Duplicate Project window will open to allow the selection of the method used to rename the components of the project.




You can select between Renumber, Rename (preferred), Prefix modes and also use Manual Entry to make changes, if desired.

### *Users Button*

Use this button to specify which users and companies should have the current project's triggers start automatically. This feature works in conjunction with the Users Button on the Trigger Setup window. Once clicked the Start Project Triggers Automatically on Login for Users window will open.




You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

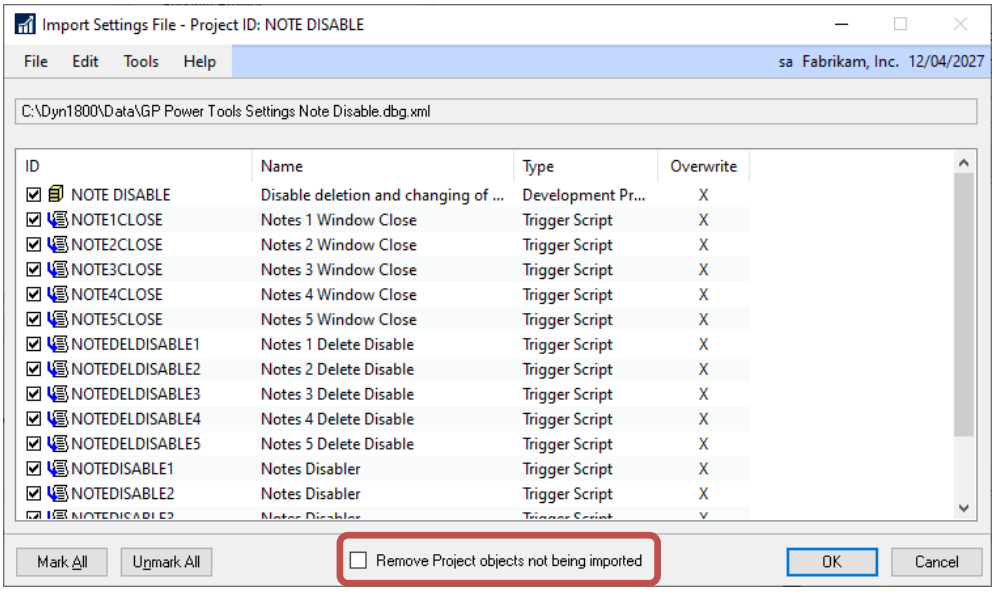
The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the project triggers should not be activate.

### *Export Button*

This button will export all the settings linked to the current project to the configuration file path specified.

### *Import Button*

This button will import from the configuration file path specified. You will have an option to Remove Project objects not being imported so you can remove objects that are no longer required when importing a project.




### *Start Button*

This button will start all triggers linked to the current project, all automatic start triggers, or the selected triggers in the list. It can also open the Trigger Status window.

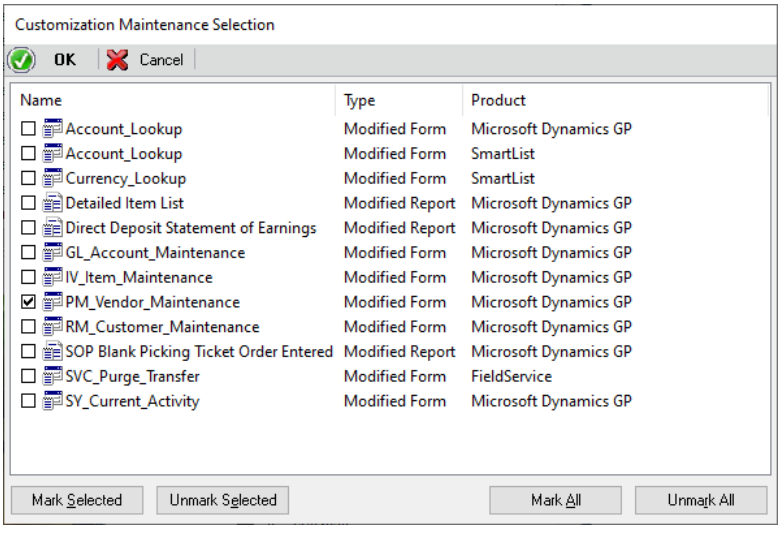
### *Stop Button*

This button will stop all triggers linked to the current project, all automatic start triggers, or the selected triggers in the list. It can also open the Trigger Status window.

### *Add Button*

This button will allow you to quickly add new Triggers, Scripts, Snippets, Parameter Lists and Messages to the current project. It can also open the Trigger Status window.

To add or remove resources from Customization Maintenance use the Customization Maintenance Selection window



*Update Triggers/Scripts Button*

Use this button to change settings on resources within the project without needing to open the resources. You can make changes to all resources or just to the selected resources in the Project Component List.

*Execute Button*

This button can be used to execute the currently selected script in the Project Component List without needing to open the script first.

*Open Form or Report Button*

This button will attempt to open the form or report associated with the current selection in the Project Component List.

*Reset Path Button*

This button will reset the Configuration File Path field back to its default value for the system. You can then adjust the path as desired.

*Redisplay Button*

This button will refresh the Project Component List.

The following is a description of the Options menu available:

*Save and Continue*

Use this menu option to save the current project without clearing the window. Control-S can be used as a shortcut.

*Execute*

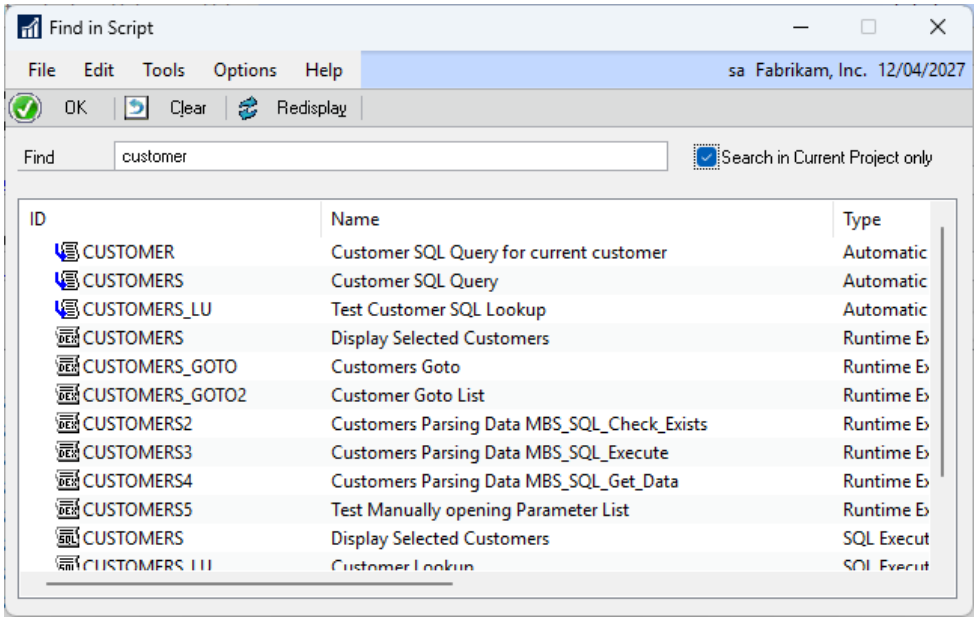
Use this menu option to save the selected script without needing to open it first. Control-E can be used as a shortcut.

*Export Compatibility Warning*

Use this menu option to disable or enable the Export Compatibility Warning when exporting projects. This warning checks that a project is compatible with Build 28 or earlier which only allowed IDs of 15 characters and Descriptions of 60 characters. Build 29 or later allows IDs of 25 characters and Descriptions of 100 characters. This setting is stored using the MBS_Debug_ExportCompatibilityWarning Dex.ini file setting.

*Find in Scripts*

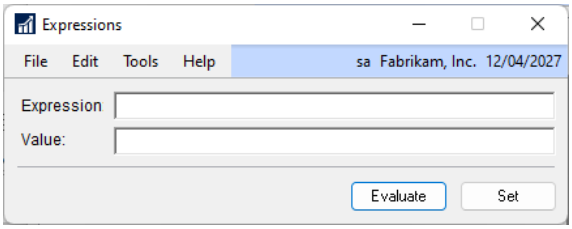
Use this menu option to open the Find in Scripts window. Use this window to search for the entered string in any script in the current project or across the entire system.




Double click on a line to open the window and script editor for that record. This will then use the Find option on the target window to locate the desired text within the script.

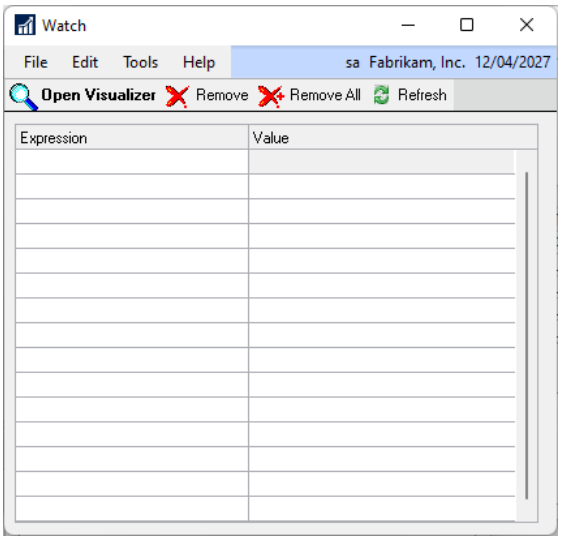
### *Debug Expressions*

Use this menu option to open the Script Debugger Expressions window. When the Expressions window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



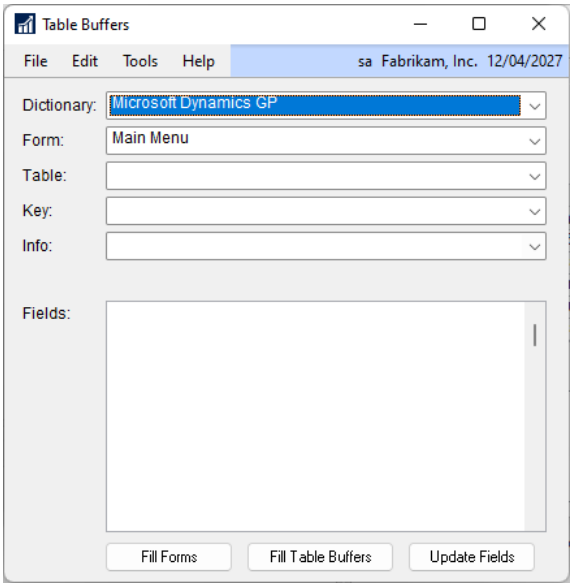
### *Debug Watch*

Use this menu option to open the Script Debugger Watch window. When the Watch window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.




### *Debug Table Buffers*

Use this menu option to open the Script Debugger Table Buffers window.



# Automatic Trigger Mode

Automatic Trigger Mode uses the logging options and Dexterity triggers to log application and SQL activity up to a specific event and exception condition. GP Power Tools can look for multiple issues.

## Introduction

The Automatic Trigger Mode of GP Power Tools came about as a result of a specific support incident. The Dynamics support team was assisting a customer with a situation that produced invalid data in a table, but no cause could be replicated. Looking at the customer's data it was verified that there was an incorrect value in the table. No one was able to identify when the previously correct value in the table was being changed to the incorrect value. GP Power Tools was used to monitor the table field in question and log the steps which led up to the field changing to the incorrect value. It was able to identify the situation and provide the exact scripts being executed up to the point the exception occurred. This information allowed the code issue to be identified and fixed.

## How to Setup

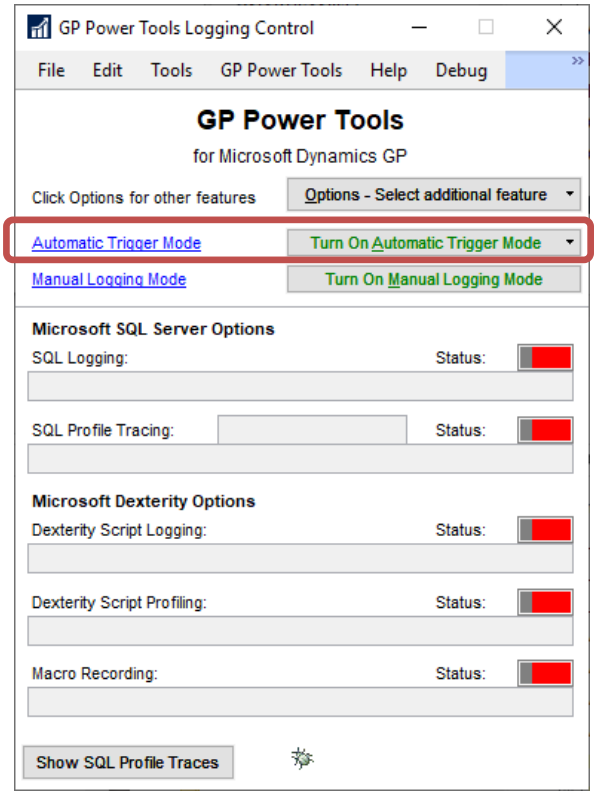
To use Automatic Trigger Mode, you must create a trigger ID using the Trigger Setup window for each issue or exception condition being monitored. For each trigger ID, an event must be identified which can be used to look for the exception condition. For example, if the exception condition involves data in a table, the trigger event used could be when the table in question is saved. If the exception condition involves a field on a window, the trigger event could be when the field in question is changed.

After the trigger event is selected, a conditional script is written using Dexterity sanScript to check whether the exception condition has actually occurred. Scripts written for this purpose will require the assistance of an experienced Dexterity developer.

Finally, the actions to perform when the exception condition occurs are defined. The trigger ID can be marked to start automatically. When the Start Trigger Automatically on Login checkbox is selected, it is possible to limit the Trigger ID to only automatically start for specified users and/or companies as well as a specified date range.

## Registration

When Automatic Trigger Mode is started either manually from the GP Power Tools main window or automatically on login, GP Power Tools registers Dexterity triggers based on the trigger IDs being activated. Once the triggers are registered all logging options are activated. GP Power Tools then waits for one of the triggers to fire.

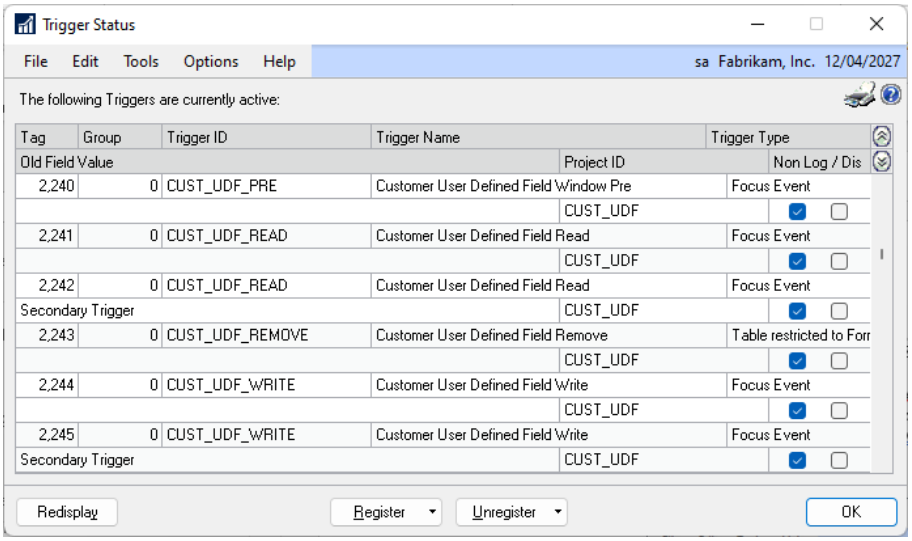



When manually activating the Automatic Trigger Mode, you can select whether to activate:

- just the DEFAULT trigger ID only,
- the logging trigger IDs marked to start automatically only,
- all logging trigger IDs in the system (except those marked as disabled),
- the non-logging trigger IDs marked to start automatically only,
- all non-logging trigger IDs in the system (except those marked as disabled), or
- all triggers for the selected project. Once selected the Project Lookup window opens to select a project to start.

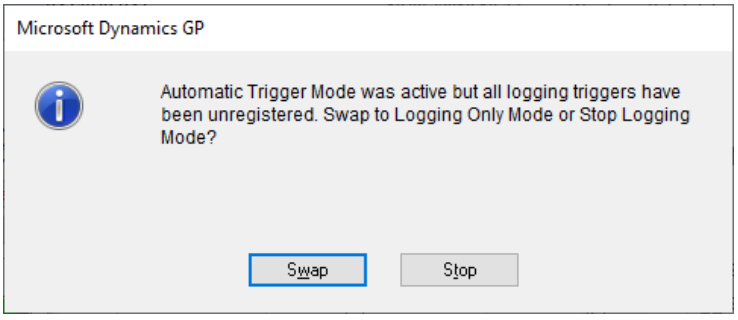
Non-logging triggers are triggers that can be registered to perform actions independently of the normal Automatic Trigger Mode triggers. They will not activate Automatic Trigger Mode and will not start the system logging. Non-logging triggers can be used to store system values prior to other triggers or used to prototype possible changes to fix an issue without the creation of a Dexterity chunk-based trigger.

Clicking on the Automatic Trigger Mode hyperlink will open the Trigger Status window which displays the Dexterity triggers are currently registered by GP Power Tools. If the trigger needs to store a previous value for a field, it will also be shown on this window.




You can also open the Trigger Status window by selecting Trigger Status from the Inquiry section of the GP Power Tools Area Page or by selecting Scripting >> Trigger Status from the Options button drop list on the main window.

From the Trigger Status window you can use the Unregister button to unregister single or multiple triggers of either the logging or non-logging type. If all logging triggers are disabled, you will be presented with a dialog providing the options to swap to logging only mode stop logging entirely.



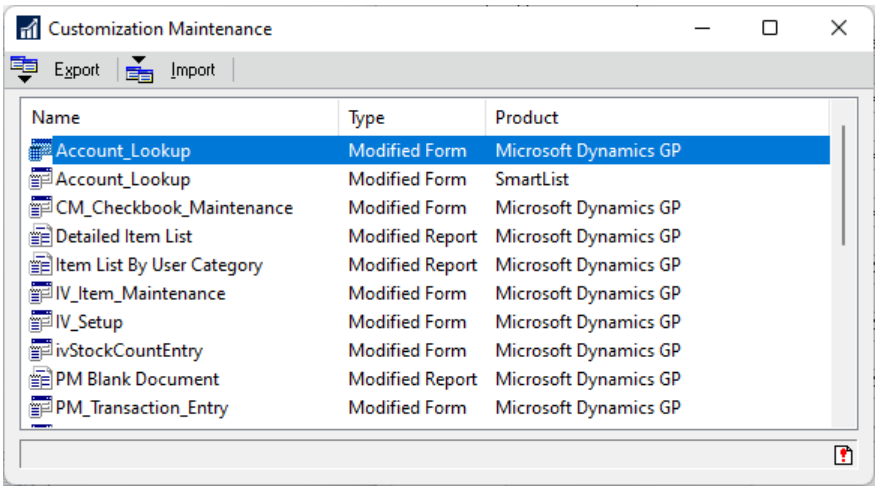

You can also use the Register button to start logging or non-logging trigger. This button functions the same as the Turn On Automatic Debugger Mode button on the previous window.

*A report of currently registered triggers can be printed using the print button on the top right of the Trigger Status window.*

The following is a description of the Options menu available:

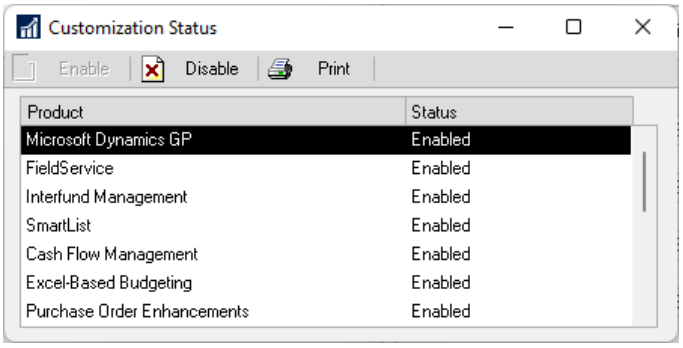
### *Customization Maintenance*

Use this menu option to open the Customization Maintenance window. Having the option on this window allows it to be opened on the web client.



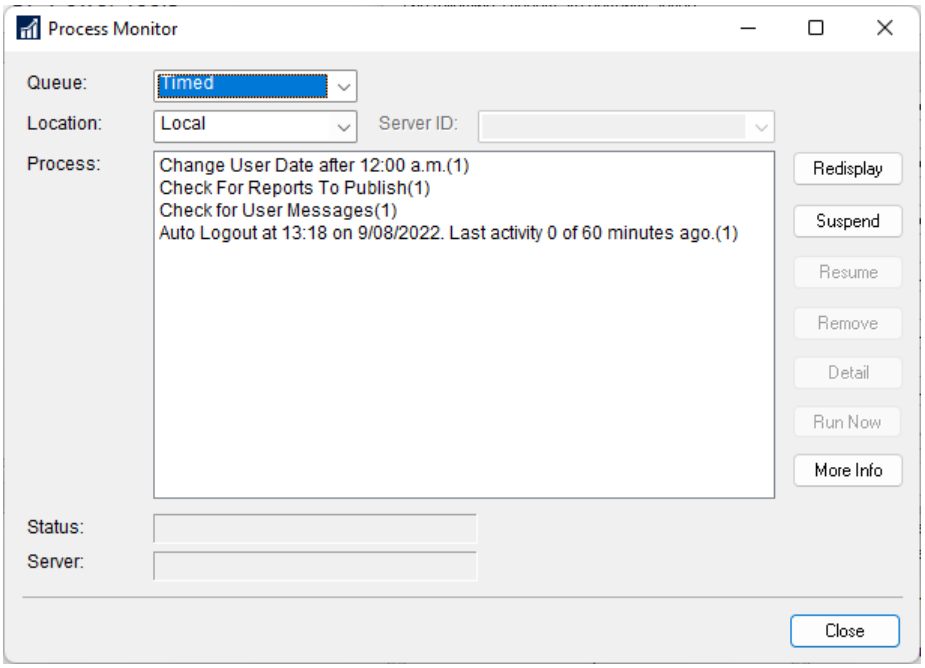

### *Customization Status*

Use this menu option to open the Customization Status window. Having the option on this window allows it to be opened on the web client.



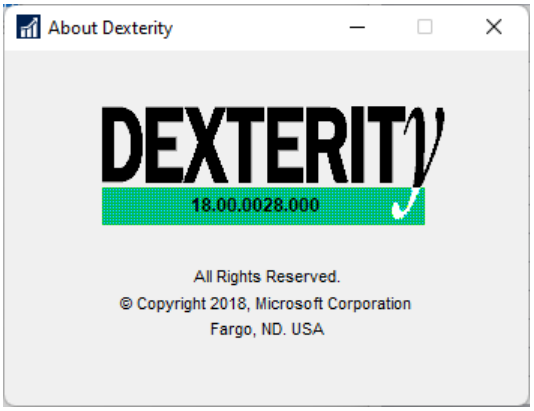
### *Process Monitor*

Use this menu option to open the Process Monitor window. Having the option on this window allows it to be opened on the web client. This window will display any background or report writer processes currently queued up in the Normal Queue. Change to the Timed Queue to see timed and scheduled processes.



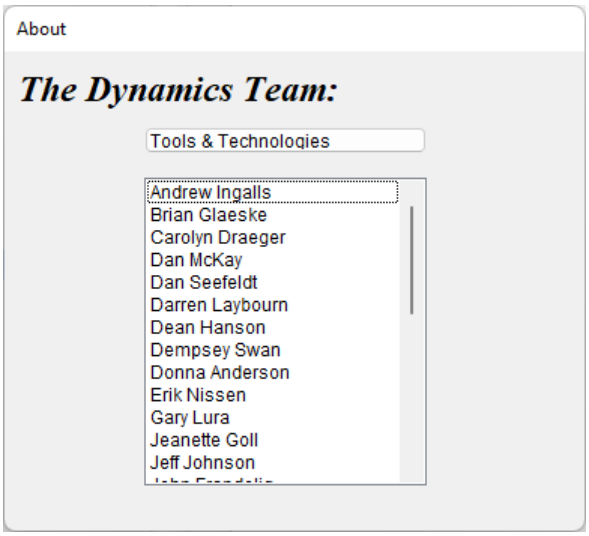

### *About Dexterity*

Use this menu option to open the About Dexterity window. This window is not normally available inside Microsoft Dynamics GP.



### *Hidden About Window*

Use this menu option to open the Hidden About window. This window is not normally available inside Microsoft Dynamics GP and shows the names of original Dexterity and Dynamics development teams. Click on the heading to change the list of people shown and click on the background to close the window.



## Triggering

When an event being monitored occurs and the Dexterity trigger is initiated or "fired" GP Power Tools looks up the trigger ID and runs the associated script to check if the exception condition has actually happened.

If the issue or exception condition is identified to have occurred by the associated script, GP Power Tools will log the results and save the log files as described in the Manual Logging Mode section. GP Power Tools then restarts the logging and continues to wait for the next trigger to fire.

If the actions to export the table record or the entire table were selected, the following files will be created:

- Record_<User>_<Company>_<Date>_<Time>.xml

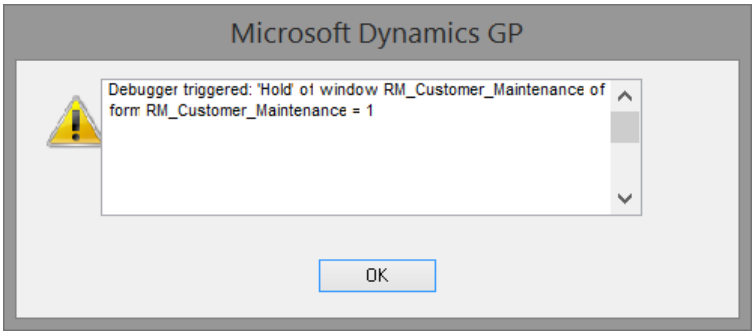
  *This file will contain the exported table record.*

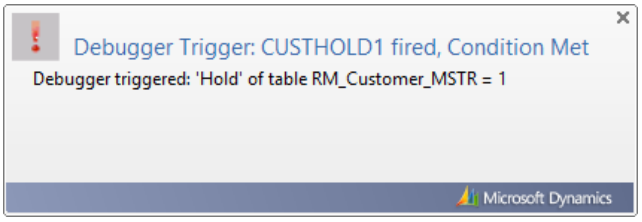
- Table_<User>_<Company>_<Date>_<Time>.xml

  *This file will contain the exported records for the entire table.*

These export files can be found in the folder where GP Power Tools is storing its data files. The default location is the data subfolder beneath the Microsoft Dynamics GP application folder. The location can be changed from the default path using the Pathname location for Debugger Setup files, exports and logs option on the Dex.ini Settings windows (see section the previous chapter).

If the action to display a message or desktop alert was selected, a dialog and/or alert with the display message specified will be shown.

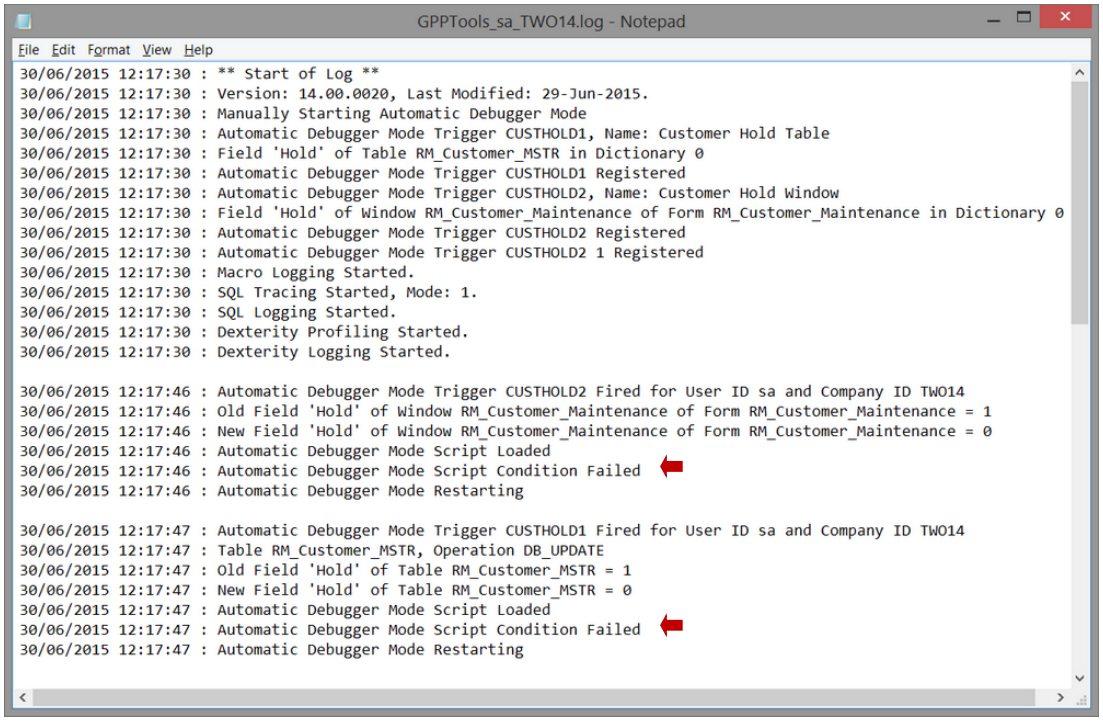


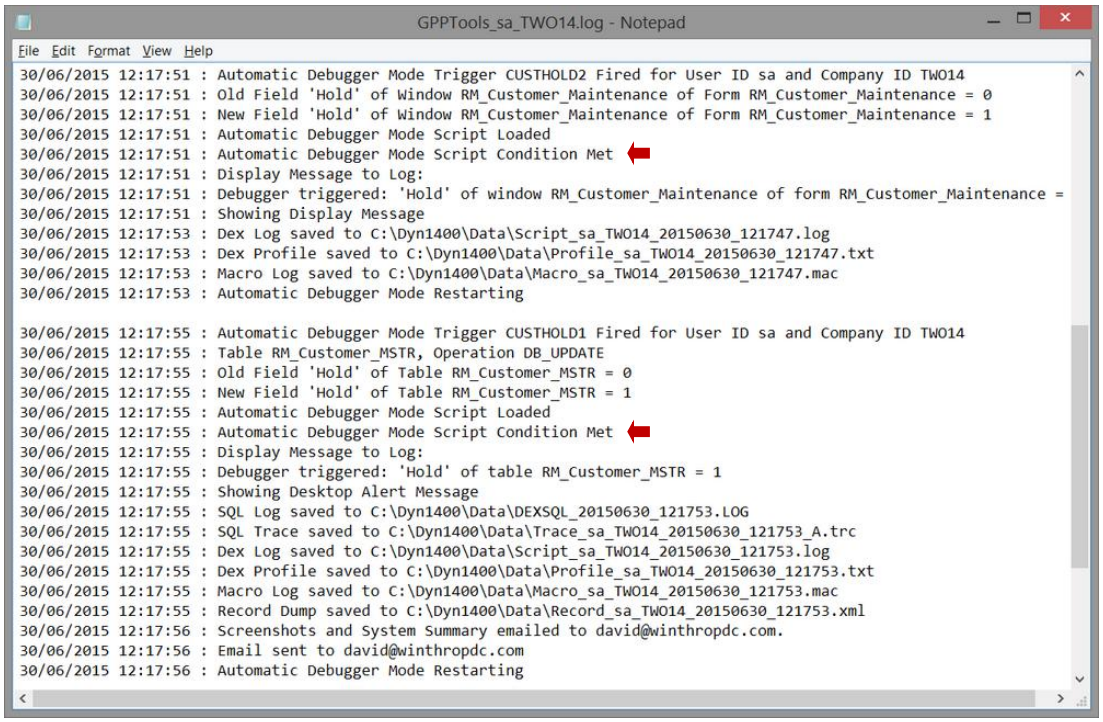

If the exception condition has not occurred, then GP Power Tools resets and continues to wait for the next trigger to fire.

## Log File

All actions by GP Power Tools are logged in the GPPTools_<User>_<Company>.log file. Below is an example log when the conditions failed.



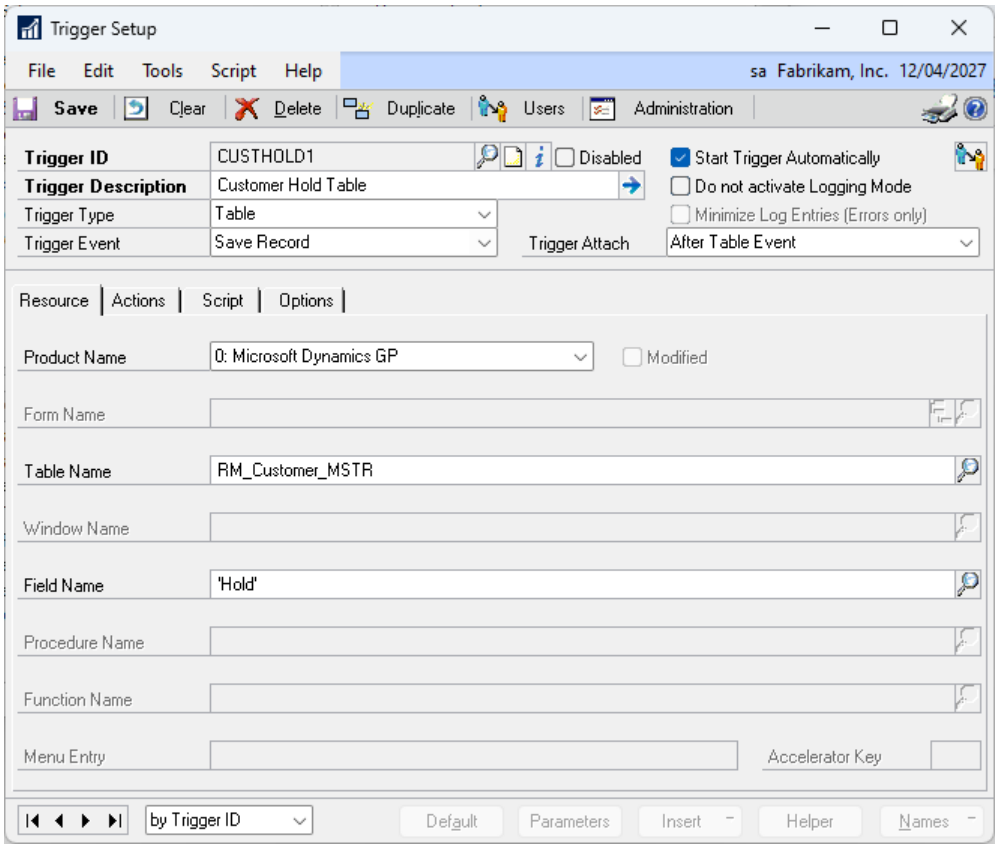

Below is an example log when the conditions were met.



## Trigger Setup

You can open the Trigger Setup window by selecting Trigger Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Trigger Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The Trigger Setup window is used to define the Dexterity triggers that will be used to look for the exception conditions.




The window is divided into a header section and four tabs; the Resource Tab, the Actions Tab, the Script Tab and the Options Tab.

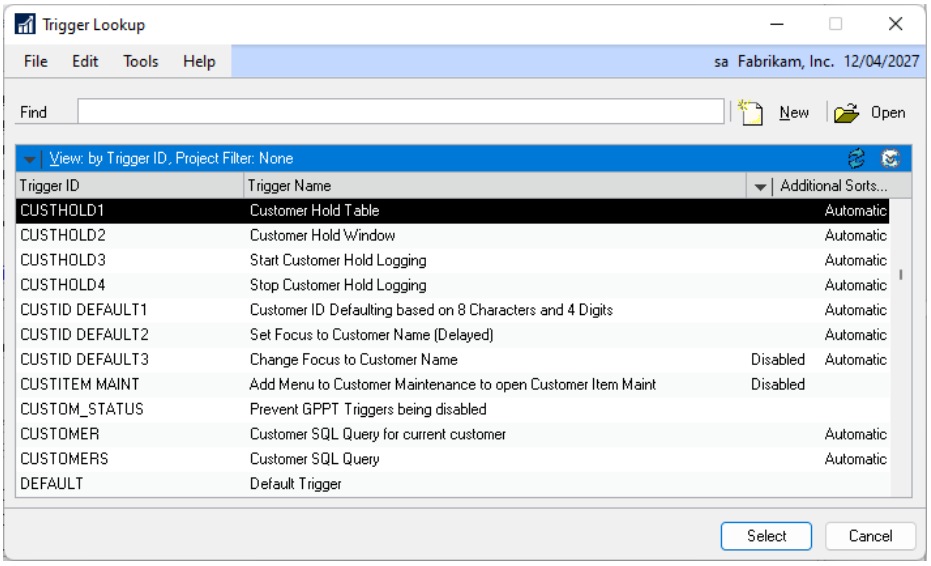
*The system will always have a trigger ID named DEFAULT. If this trigger ID is deleted, it will be added back automatically as a blank trigger ID. The use of this trigger ID is optional.*

When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

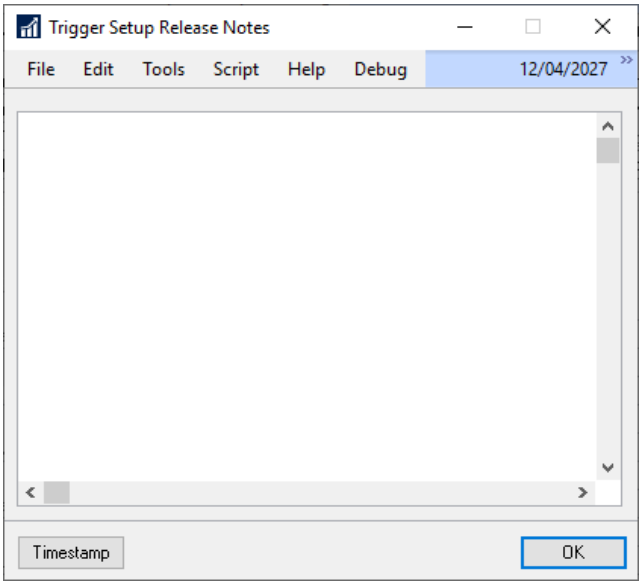
Below is a description of the individual header fields on the window:

*Trigger ID*

This field contains a unique identifier for each trigger in the system. The lookup button can be clicked to select from existing trigger IDs. The lookup will be filtered to the current project if the trigger belongs to a project.



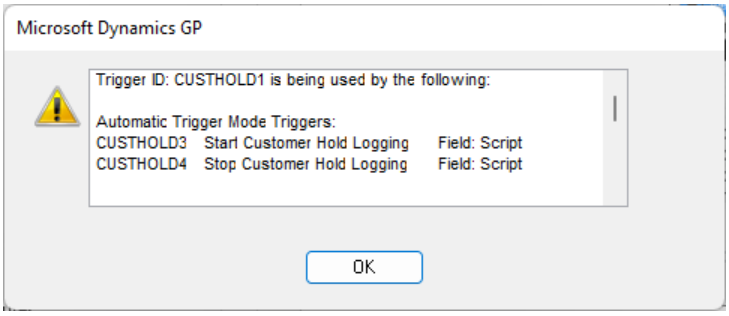

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Trigger IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*Trigger Information*

Click this button to display what resources are using this trigger.

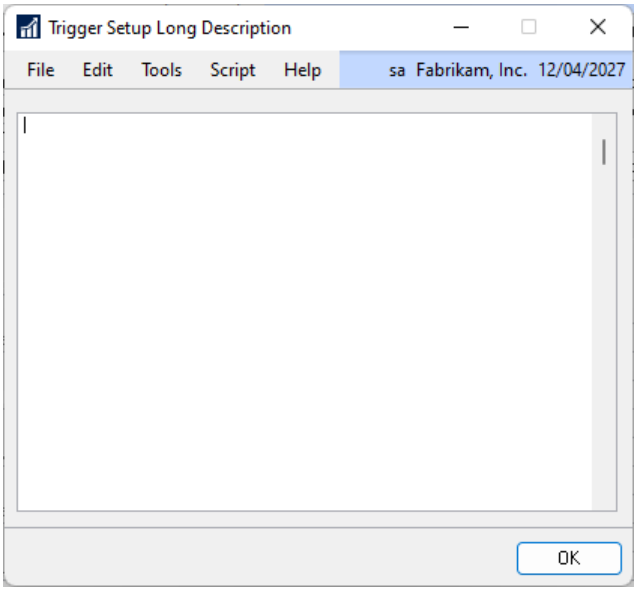



*Trigger Description*

This field contains a description for the trigger.

*Long Description*

Use the Trigger Description expansion button to open the Long Description window. Use this field for a more detailed description of the trigger.




*Trigger Type*

This drop-down list specifies the type of trigger being defined. The following objects can be selected.

- Table
- Table restricted to Form
- Procedure
- Function
- Focus Event
- Focus Event with Table
- Warning Dialog
- Timed Event
- Form Level Menu
- Field Context Menu
- Login/Logout Event
- Scheduled Event
- Application Level Menu

*Trigger Event*

This drop-down list specifies the event for the selected object. The following events can be selected depending on the trigger type selected:

- Table
  - Save Record
  - Delete Record
  - Read Record
- Table restricted to Form
  - Save Record
  - Delete Record
  - Read Record
- Procedure
  - Global Level
  - Form Level
  - Global Level with Parameters
  - Form Level with Parameters
- Function
  - Global Level
  - Form Level
  - Global Level with Parameters
  - Form Level with Parameters
- Focus Event
  - Form Pre
  - Form Post
  - Window Pre
  - Window Post
  - Window Activate
  - Scroll Fill
  - Scroll Pre
  - Scroll Change
  - Scroll Post
  - Scroll Insert
  - Scroll Delete
  - Field Pre
  - Field Change
  - Field Post
  - Field Value Changed
  - Modal Dialog
  - Context Menu
- Focus Event with Table

Form Pre
Form Post
Window Pre
Window Post
Window Print
Window Activate
Scroll Fill
Scroll Pre
Scroll Change
Scroll Post
Scroll Insert
Scroll Delete
Field Pre
Field Change
Field Post
Field Changed
Modal Dialog
Context Menu

- Warning Dialog
  - Warning Dialog
- Timed Event
  - Every 1 Minute
  - Every 5 Minutes
  - Every 10 Minutes
  - Every 15 Minutes
  - Every 30 Minutes
  - Every 60 Minutes
- Form Level Menu
  - Form Level
- Field Context Menu
  - Field Context
- Login/Logout Event
  - Login Event
  - Logout Event
  - Starting Triggers
- Scheduled Event
  - Daily Event
  - Weekly Event
  - Monthly Event
- Application Level Menu
  - Add Menu to Top
  - Add Menu to Bottom
  - Add Menu Below Entry

*While GP Power Tools can trigger against global and form level procedures and functions, it is unable to obtain the parameter lists for those procedures and functions.*

*To use the Application Level Menu Trigger Type, the Visual Studio Integration Toolkit product must be installed.*

*Trigger Attach*

This drop-down list specifies when the code for the Dexterity trigger is run when the selected event for the selected object occurs. The following attach modes can be selected depending on the trigger type selected:

- Table
  - After Table Event
- Table restricted to Form
  - After Table Event
- Procedure
  - Before Original
  - After Original
- Function
  - Before Original
  - After Original
- Focus Event
  - Before Original
  - After Original
  - After Original Delayed
- Focus Event with Table
  - Before Original
  - After Original
- Warning Dialog
  - Before Original
  - After Original
- Timed Event
  - After Timed Event
- Form Level Menu
  - After Menu Selected
- Field Context Menu
  - After Menu Selected
- Login/Logout Event or Starting Triggers
  - After Login Event
  - After Login Event (Delayed)
  - After Login Event (Background)
  - After Login Event (After Background)
  - Before Logout Event
  - After Starting Triggers
- Scheduled Event
  - After Logging In
  - After Time XX:XX
  - After Login on DOW
  - After Login on Day X
- Application Level Menu
  - After Menu Selected

*When using table trigger type, it is possible to trigger only after a successful table event. This means this option cannot be used to capture a failed save, delete, or read event.*

*Disabled*

When this checkbox is marked, the current trigger is disabled and will never be activated.

*Start Trigger Automatically on Login*

When this checkbox is marked, the current trigger will be activated automatically after logging into Microsoft Dynamics GP. Use the Users button to specify the individual user and companies to limit for whom the trigger is used.

*Do not activate Logging Mode*

When this checkbox is marked, the trigger will not start logging and will not activate the Automatic Trigger Mode. It allows a trigger to be registered and used without the overhead of maintaining the log files.

*Non-logging triggers can be started automatically on login or started manually from the Automatic Trigger Mode Turn On button. To stop a non-logging trigger, use the Unregister button on the Trigger Status window.*

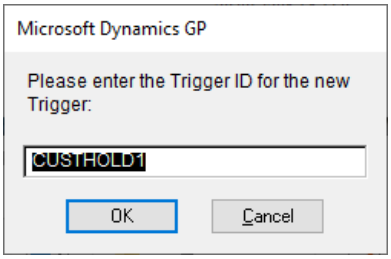
*Minimize Log Entries*

When using a Non-logging trigger, this option can be enabled to prevent the trigger generating entries in the GPPTools_<User>_<Company>.log file unless an error occurs.

The following is a description of the additional buttons on the window:

*Duplicate Button*

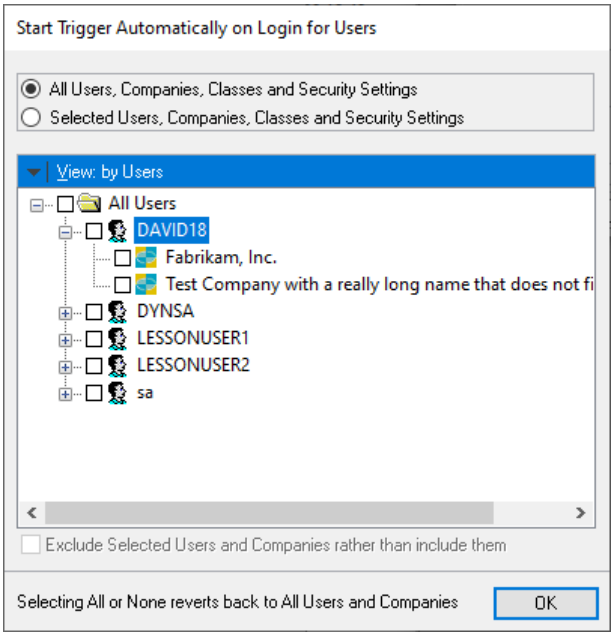
Use this button to duplicate or rename the current trigger ID and create a new trigger ID. This is useful when an existing trigger ID is very similar to the new one you want to create.




A new trigger ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies should have the current trigger start automatically. Once clicked the Start Trigger Automatically on Login for Users window will open.



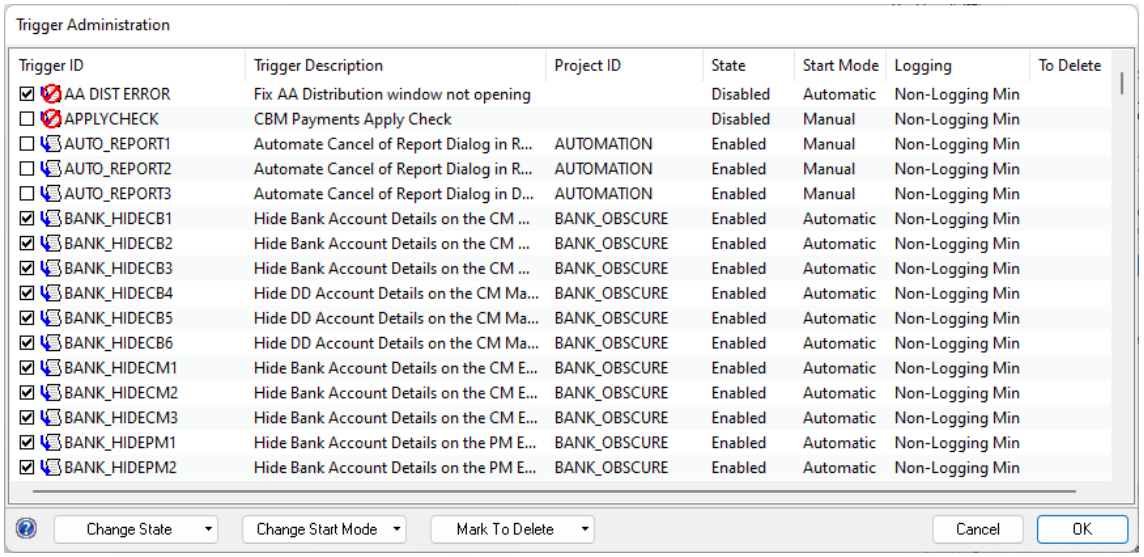

You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the trigger should not be activate.

### *Administration Button*

Use this button to administer multiple Automatic Trigger Mode Trigger IDs at the same time. Once clicked the Trigger Administration window will open.




*When the Trigger Administration window is opened, the current Trigger ID is saved automatically. The Trigger Administration window is modal and must be closed before continuing to use other windows.*

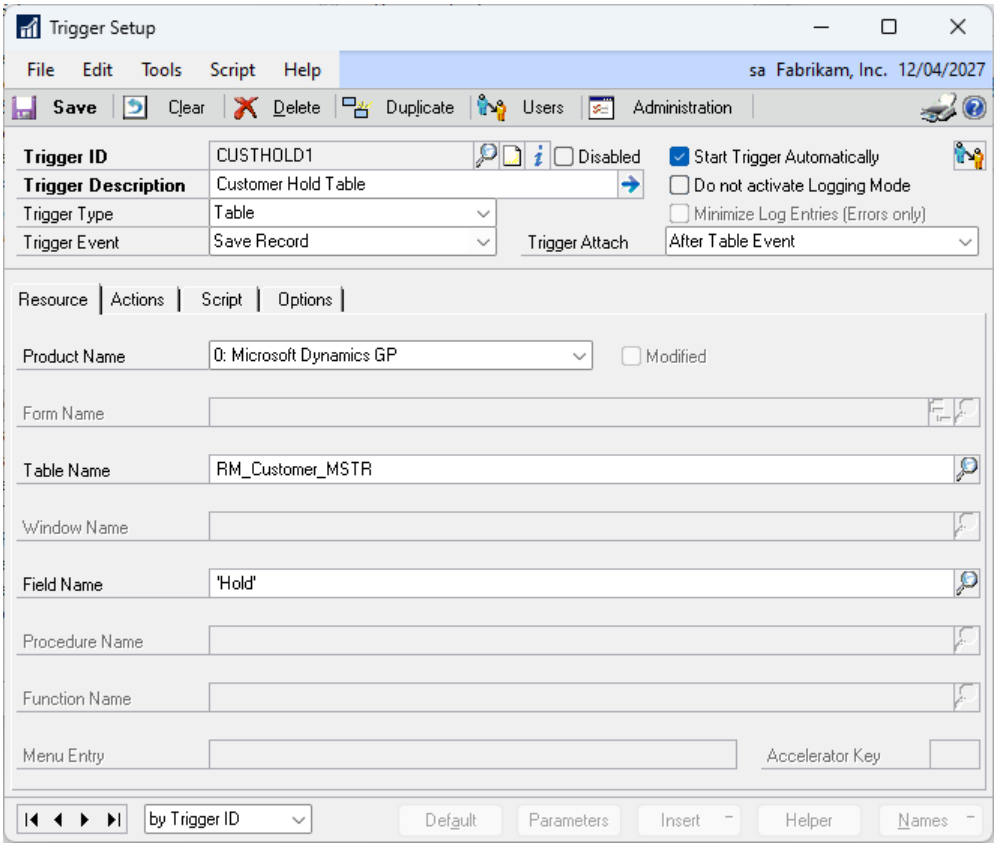
The window shows the current status of the Trigger IDs in the system. Triggers can be Enabled or Disabled, have their start mode changed between Manual and Automatic, or be deleted in bulk from this window.

To make changes, select the Trigger IDs (use control and shift keys to multi-select) and use the Change State, Change Start Mode, and Mark To Delete Buttons.

The selected changes will be made when OK is clicked. Clicking Cancel will close the window without applying any pending changes.

## Resource Tab

The Resource tab contains the definition of the resource to apply the trigger against.




The following is a description of the individual resource selection fields on the tab. The actual fields available depend on the settings for Trigger Type and Trigger Event fields. The lookup button can be used to open the Form Explorer or the Table Explorer to select the required resource:

*Product Name*

This drop-down list contains a list of products currently installed on the Microsoft Dynamics GP workstation.

*Modified*

This checkbox can be used to force the trigger register in the context of the modified dictionary. This allows the trigger to reference Modifier added local fields.

*To be able to register triggers against modified dictionaries, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

*Form Name*

This field contains the technical name for the form selected.

*Table Name*

This field contains the technical name for the table selected.

*Window Name*
This field contains the technical name for the window selected.

*Field Name*
This field contains the technical name for the field selected.

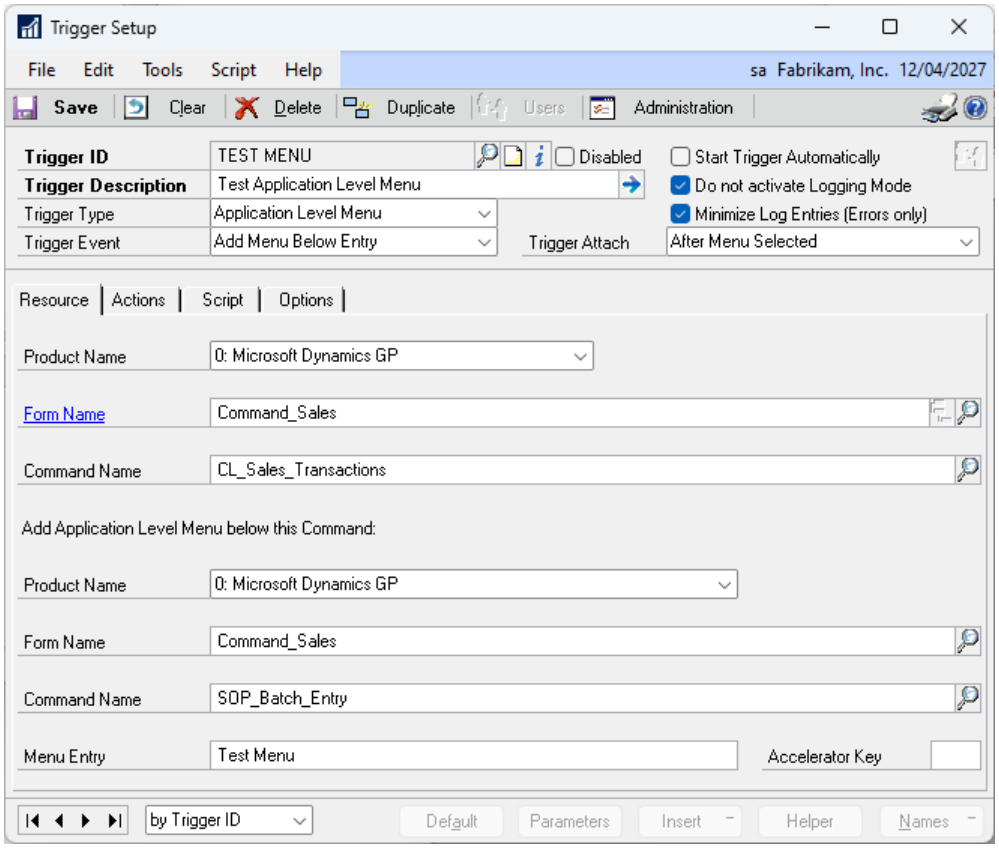
*Procedure Name*
This field contains the technical name for the procedure selected.

*Function Name*
This field contains the technical name for the function selected.

*Menu Entry*
This field contains the description to be displayed on the Form Menu created by this trigger.

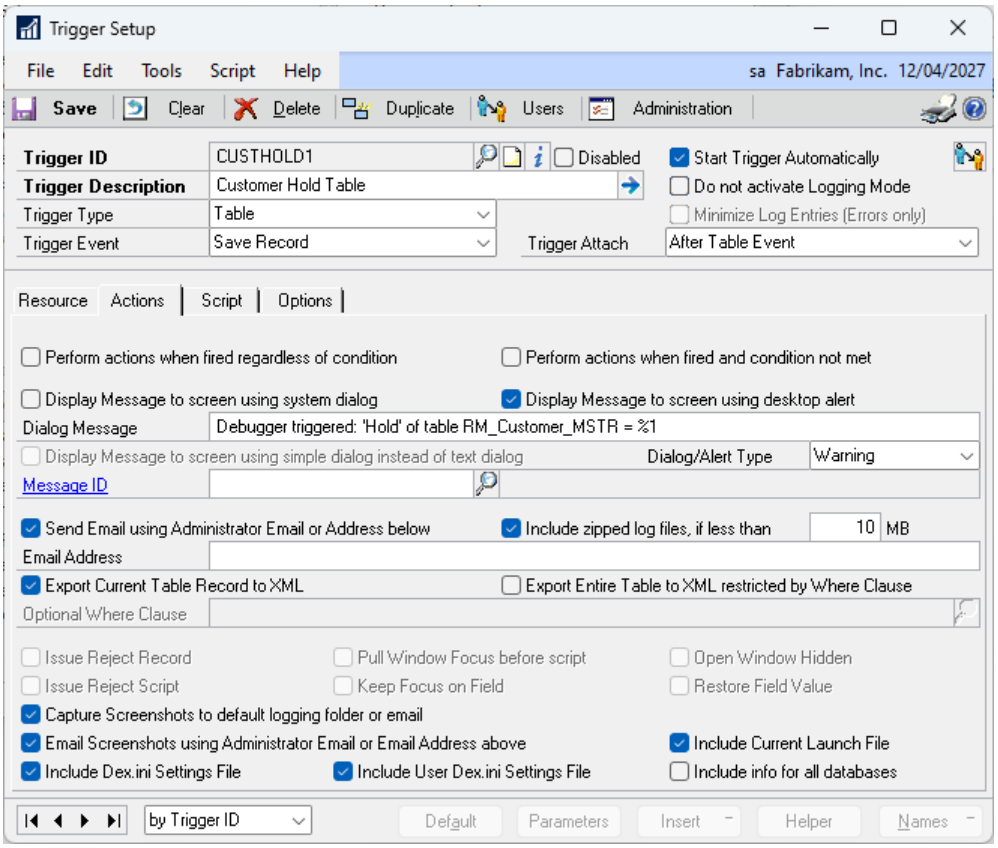
*Accelerator Key*
This field contains an optional accelerator shortcut key (used with Control) for the menu entry.

The Resource tab changes when the Trigger Type of Application Level Menu is selected so you can enter the one or two sets of information need to specify the menu commands.



## Actions Tab

The Actions tab contains the actions to perform when the trigger has fired.




The following is a description of the individual action fields on the tab. These actions will be processed when the trigger fires and the conditional script returns true or if the Perform actions when fired regardless of condition checkbox is selected.

*Perform actions when fired regardless of condition*

Check this checkbox when you want the actions to be processed when the trigger is fired regardless of whether the conditional script returns true.

*Perform actions when fired and condition not met*

Check this checkbox when you want the actions to be processed when the trigger is fired and the conditional script returns false.

*Display Message to screen using system dialog*

Select this checkbox if you want the message displayed to the screen in a system dialog box.

*Display Message to screen using desktop alert*

Select this checkbox if you want the message displayed to the screen in a desktop alert.

*Dialog Message*

This field contains the message which will be logged and displayed if the Display Message checkbox is selected. When the Field Name is specified, the message can contain the %1 placeholder which will be substituted with the field value when the message is displayed.

The message can be programmatically overridden by using the MBS_Triggerl_Update_Dialog Helper Function the Trigger handler script.

*Display Message to screen using simple system dialog instead of text box dialog*

Select this checkbox if you want the message displayed to the screen in a simple system dialog instead of a text box dialog.

*Dialog/Alert Type*

Use this drop-down list to select between Information, Warning (default), Error and Debug dialogs and desktop alerts. Debug dialogs and desktop alerts are only shown when the Debug menu is enabled and Show Debug Messages is enabled. These Debug settings can be changed on the Dex.ini Settings window.

*Message ID*

Use this field to define a Message ID to be used instead of the default Dialog Message. Messages have the advantage of only being defined once and can automatically change depending on the language of the system. To setup Messages use the Messages Setup window.

*Send Email using Administrator Email or Email Address below*

When this checkbox is selected, an email with the log details of the trigger will be sent to the Administrator Email address as setup in the Administrator Settings window, or to the specified Email Address.

*Include zipped log files*

Check this option to include the captured log files in a zipped archive file in the email sent.

*If less than X MB*

Specify the maximum allowed size for the zipped archive file.

*Email Address*

This field can be used to specify an email address to use instead of the default Administrator Email.

The email address can be programmatically overridden by using the MBS_Triggerl_Update_Email Helper Function the Trigger handler script.

*Export Current Table Record to XML*

When this checkbox is selected, the current table buffer contents will be exported as an XML file. This action is only valid for Table triggers.

*Export Entire Table to XML restricted by Where Clause*

When this checkbox is selected, the entire table contents will be exported as an XML file. This action is only valid for Table triggers. A SQL Where Clause can be specified to restrict the records exported.

*Optional Where Clause*

This field can be used with the Export Entire Table option to define a SQL Where Clause to restrict the records exported to XML. This field is only valid for Table triggers.

*Issue Reject Record*

When this checkbox is selected, a reject record command will be issued to prevent the current record being shown in a scrolling window. This action is only valid for Non-logging Focus Event triggers attached to the Scroll Fill Event.

*Pull Window Focus before script*

When this checkbox is selected, the focus will be pulled from the window before the trigger script is executed. This will ensure that any pending change or post scripts are executed and any fields will have the correct changed value.

*Open Window Hidden*

When this checkbox is selected, the window will open hidden (off display) and remain so until it is closed. This action is only valid for Non-logging Focus Event triggers attached to the Window Pre Event.

*Issue Reject Script*

When this checkbox is selected, a reject script command will be issued to abort the original code from executing. This action is only valid for Non-logging Focus Event triggers running before the original code.

*Keep Focus on Field*

When this checkbox is selected, the focus will be kept on the current field. This action is only valid for Non-logging Focus Event triggers running before the original code when the Reject Script option is used.

*Restore Field Value*

When this checkbox is selected, the original value of the current field will be restored. This action is only valid for Non-logging Focus Event triggers running before the original code when the Reject Script option is used.

*Capture Screenshots to default logging folder or email*

When this checkbox is selected, the ScreenShot utility will be used to capture screenshots of all open windows and either save them to the logging folder or email them.

*Email Screenshots using Administrator Email or Email Address below*

When this checkbox is selected, an email with the captured screenshots will be sent to the Administrator Email address as setup in the Administrator Settings window, or to the specified Email Address.

*Include Dex.ini Settings File*

This checkbox tells the ScreenShot utility whether to include the Global level Dex.ini settings file as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include User Dex.ini Settings File*

This checkbox tells the ScreenShot utility whether to include the User level Dex.ini settings file as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include Current Launch File*

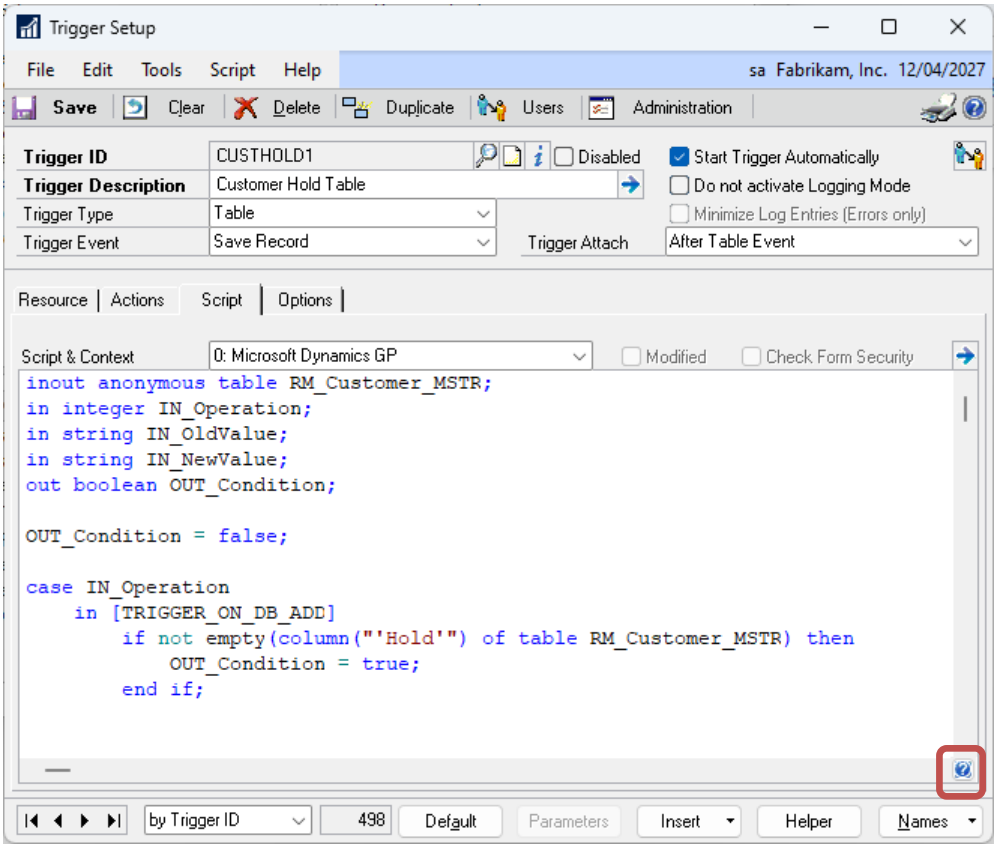
This checkbox tells the ScreenShot utility whether to include the launch file, usually Dynamics.set, as an attachment for the email. The default setting for this checkbox can be set up in the Administrator Settings window.

*Include info for all databases*

This checkbox tells the ScreenShot utility whether to include information for all databases or just the system database and current company database in the System Status report. Not including information for all databases gives better performance on systems with many companies. The default setting for this checkbox can be set up in the Administrator Settings window.

## Script Tab

The Script tab contains the Conditional script to be executed when the trigger fires.




The following is a description of the individual script fields on the tab.

### *Script Context*

This drop-down list contains a list of products currently installed on the Microsoft Dynamics GP workstation. It is used to select the dictionary context that the conditional script will be executed in. The script context is usually the same as the dictionary ID, but can be changed if the script needs to be executed in a different dictionary to where the trigger is registered.

### *Modified*

This checkbox can be used to force the script to execute in the context of the modified dictionary. This allows the script to reference Modifier added local fields. Changing this setting will automatically update the script to alter the parameter passing method used.

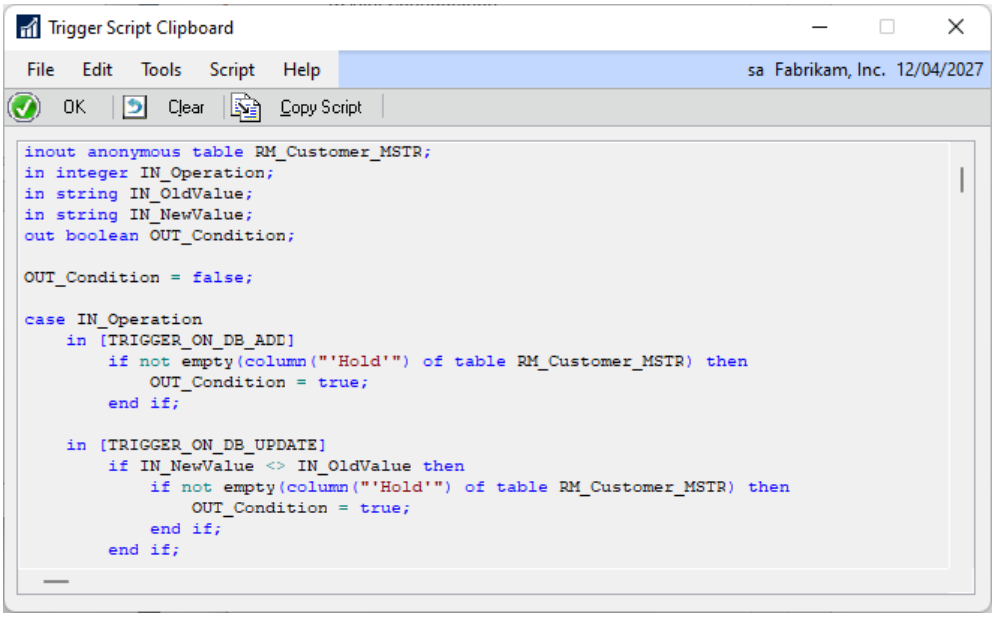
*To be able to execute scripts against modified dictionaries, the WinthropDC.GpPowerToolsVC.dll Addins must be installed.*

### *Check Form Security*

This checkbox can be control when the script is executed. If selected, the script will only run if version of the form opened matches the Script Context product and Modified checkbox.

### *Clipboard Button*

This expansion button opens the Trigger Script Clipboard window. This window can be used as temporary storage when editing scripts to allow for cutting and pasting between scripts. Use the Copy Script Button if you wish to copy the current script to the clipboard window and the Clear Script Button to clear the clipboard window contents.




*When changing the trigger settings or trigger resource, you will be asked if you want to copy to the clipboard before the script is reset. You can then copy back the portions of the script you want to keep. The clipboard window will close with the Trigger Setup window.*

### *Conditional Script*

This text field contains the script to be executed when the trigger fires. The script will be populated with a default script when the trigger type, trigger event and resource information are selected. The script will have the required parameters, including a boolean OUT_Condition. The script can be used to check for the exception condition being targeted and then set OUT_Condition to true if the condition has occurred. The script is checked for syntax errors when saved.

*Using the Helper Functions (see below), a script created in the Runtime Execute Setup window, the SQL Execute Setup window or the .Net Execute Setup window can be loaded and executed from within a conditional script of a trigger.*

The following is a description of the additional buttons on the tab:

### *Help Button*

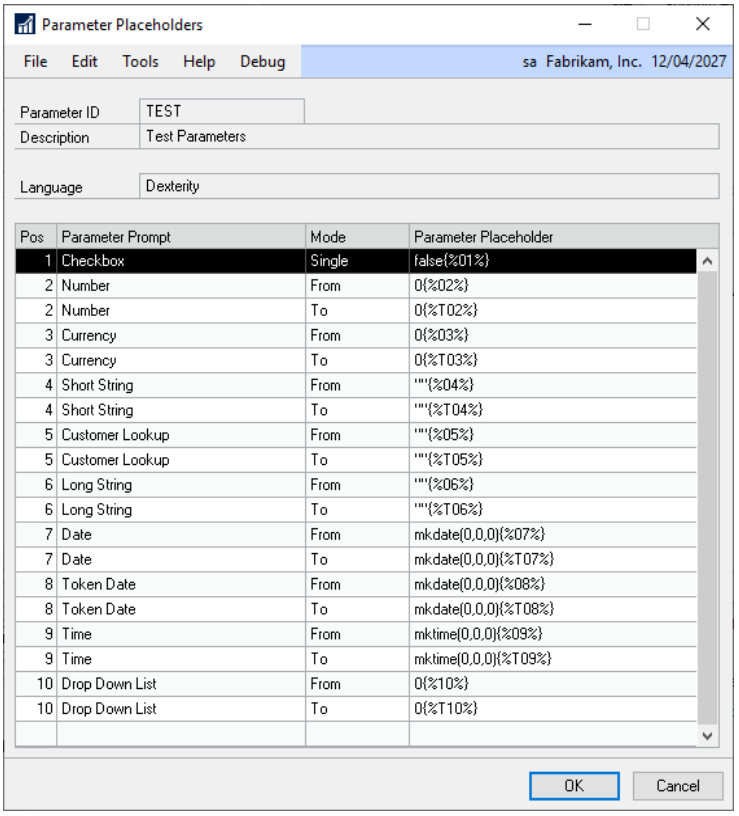
Use this button (highlighted on screenshot) to open the full Dexterity Help file.

### *Default Button*

Use this button to reset the Message and Conditional Script fields to the default settings based on the trigger and resource settings.

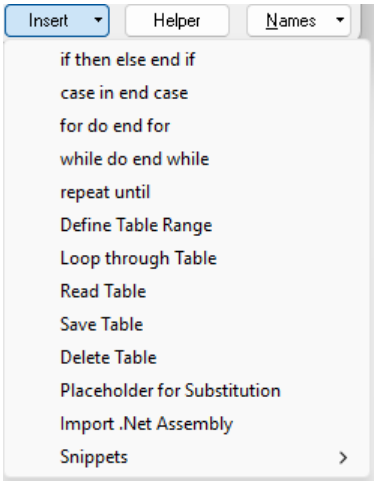
### *Parameters Button*

Use this button to insert a Parameter Placeholder into the script for the Parameter List selected with the Parameter ID on the Options tab.

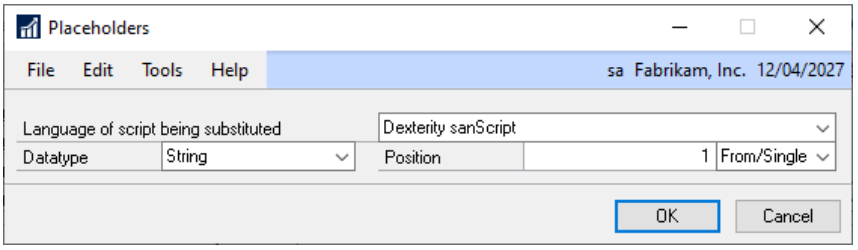


### *Insert Button*

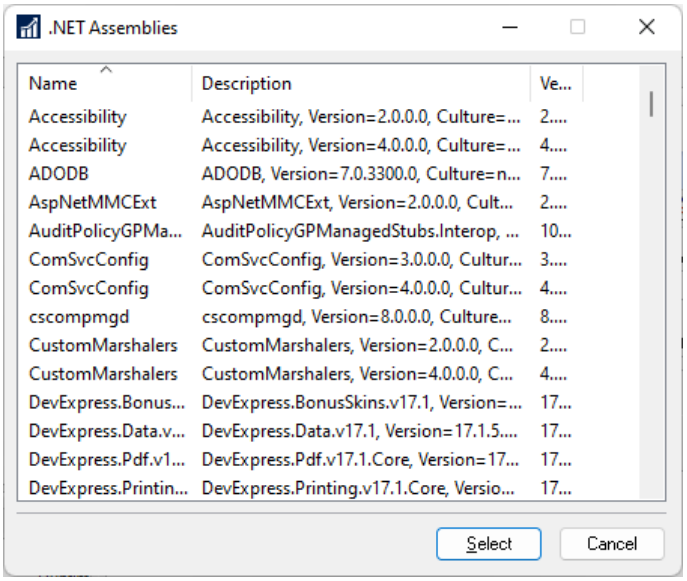
Use this button to insert a Dexterity sanScript code construct, Parameter Placeholders, .Net Assemblies or code Snippets. The available constructs are shown below:




If Placeholder for Substitution is selected, the Placeholders window will open.



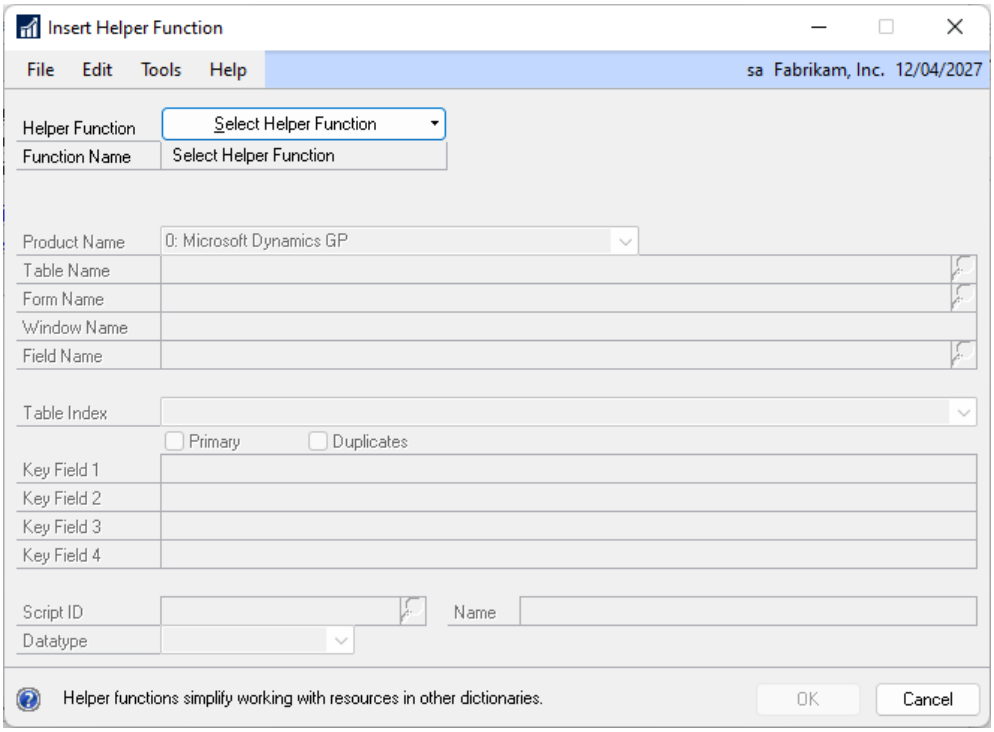

If Import .Net Assembly is selected, the .Net Assemblies window will open.



If Snippets is selected, the snippets available for the current language are displayed in their groups as specified on the Snippet Setup window.

### *Helper Button*

Use this button to open the Insert Helper Function window. The appropriate code for the selected helper function will be inserted into the script.



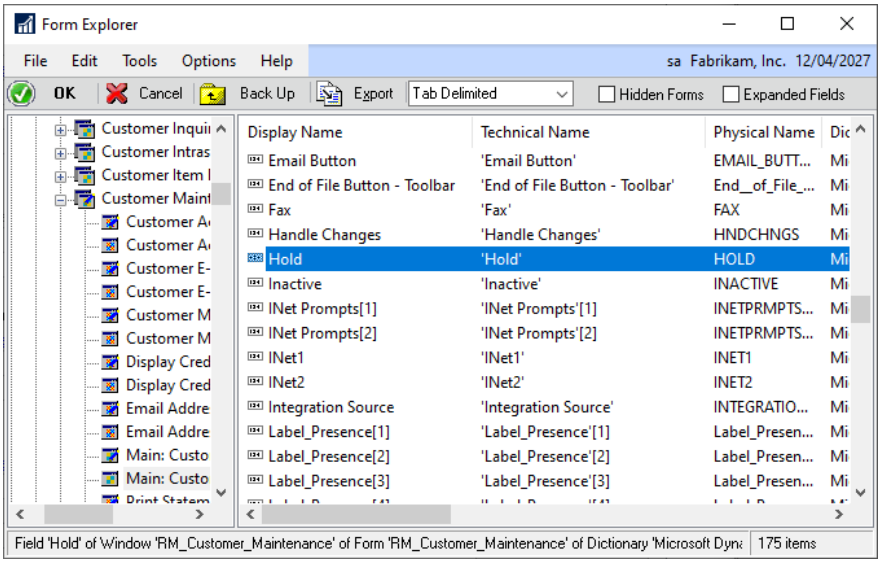

Helper functions can be used to read or write a window or table field in any window or table from any loaded dictionary. When setting a window field you can select whether to execute the field's change script. When setting a table field you can select whether adding a new record is allowed.

The table-based help functions currently support up to four key fields. The individual helper functions are covered in more detail in a later chapter.

### *Names Button*

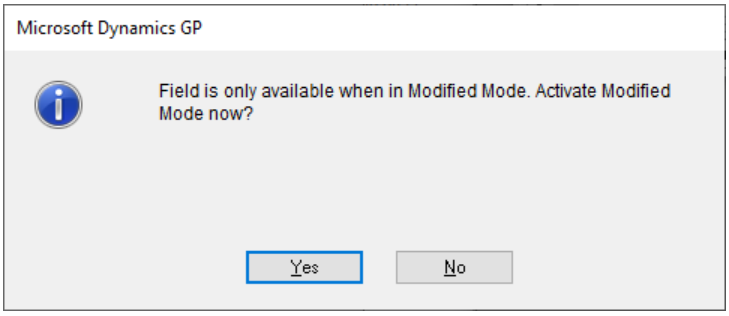
Use this button to insert a dictionary resource into the script.

Once Forms, Windows & Fields is selected the Form Explorer window will open.

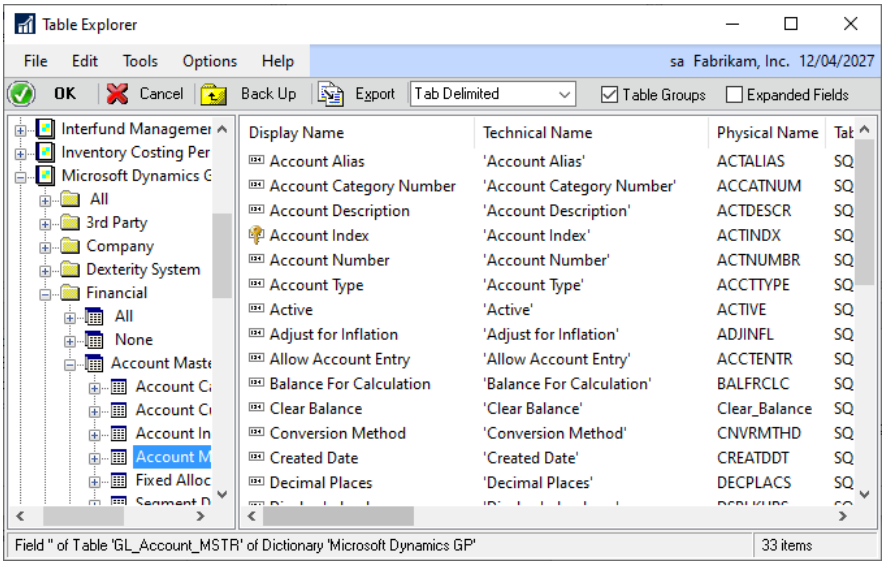



*To insert a form name or window name, select the resource in the right-hand pane and click OK. If no resources are selected on the right-hand pane, the currently selected resource in the left-hand pane will be used when OK is clicked. Multi-select is available to insert multiple resources at one time.*

If a Modified field is selected and the Modified checkbox is not selected, the following dialog will be displayed to suggest activating Modified mode now.

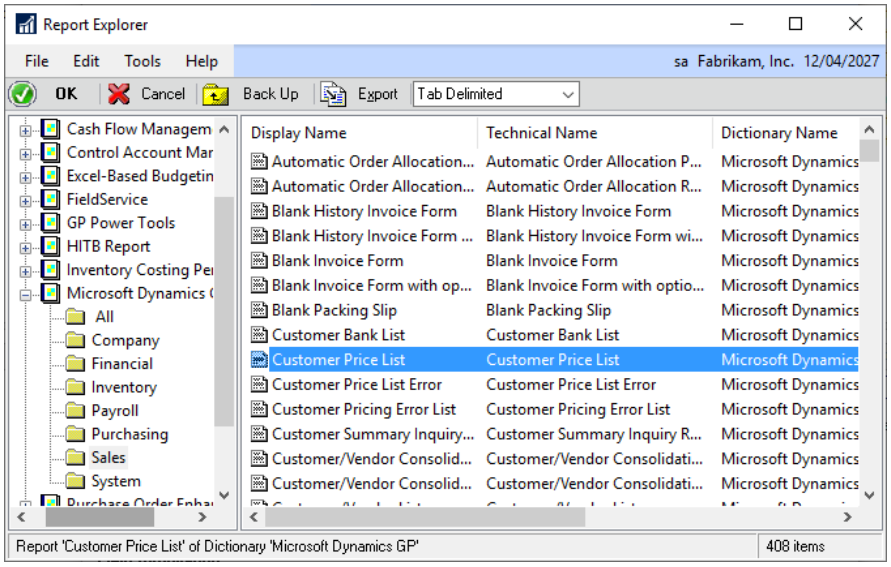


Once Tables & Fields is selected the Table Explorer window will open.



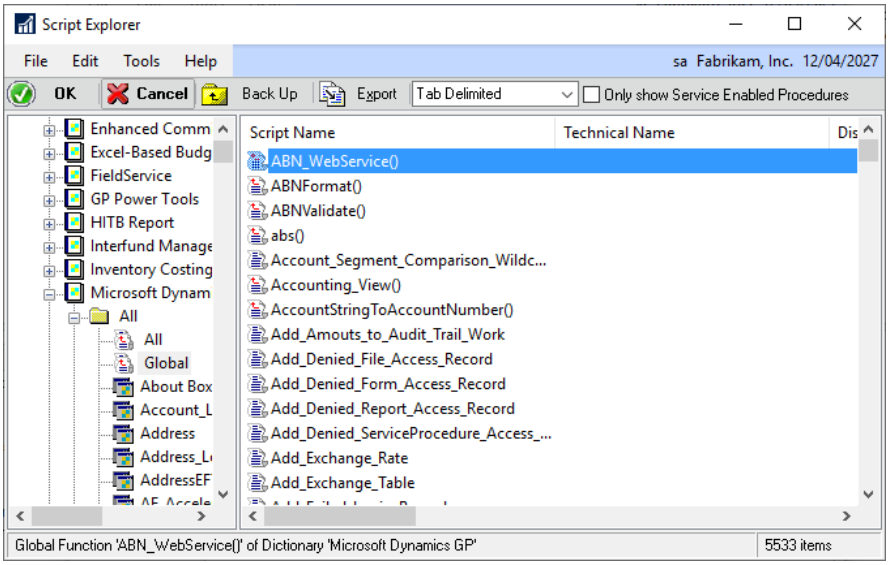

*To insert a table name, select the resource in the right-hand pane and click OK. If no resources are selected on the right-hand pane, the currently selected resource in the left-hand pane will be used when OK is clicked. Multi-select is available to insert multiple resources at one time.*

Once Reports is selected the Report Explorer window will open.




*To insert a report name, select the resource in the right-hand pane and click OK.*

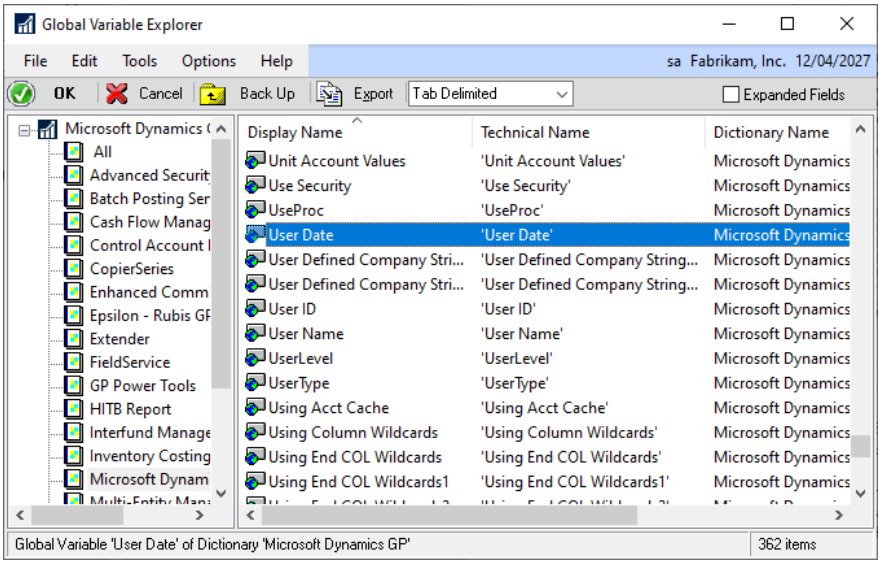
Once Procedures & Functions is selected the Script Explorer window will open.




*To insert a procedure or function name, select the resource in the right-hand pane and click OK.*

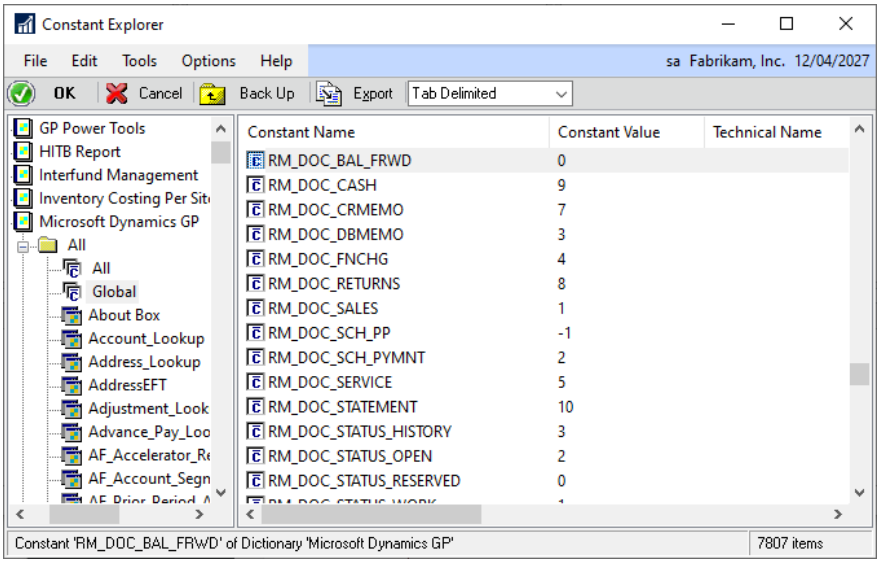
*Selecting Procedures & Functions with Parameters will allow for the full call syntax to be inserted into the script with the parameter list (if available).*

Once Global Variables is selected the Global Variable Explorer window will open.



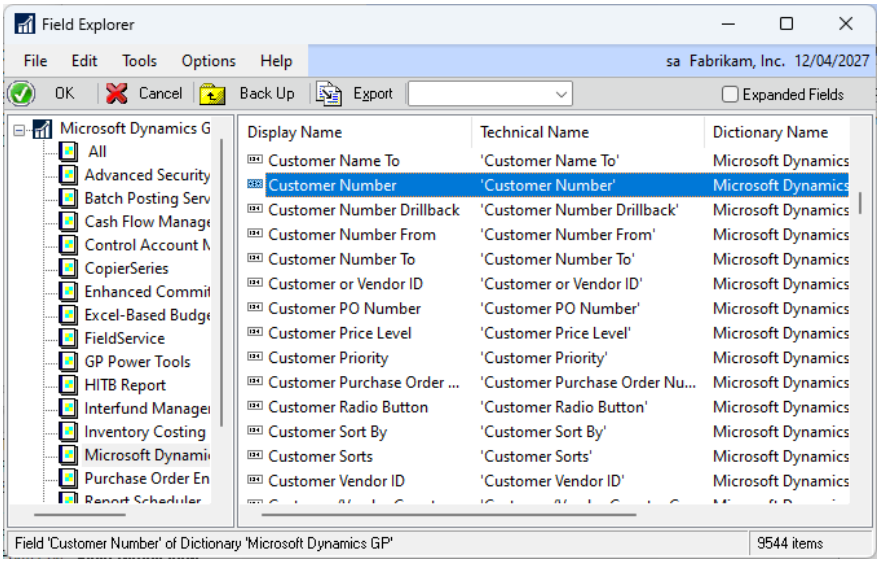

*To insert a global variable name, select the resource in the right-hand pane and click OK.*

Once Constants or Constants (value) is selected the Constant Explorer window will open.




*To insert a Constant name or Constant value, select the resource in the right-hand pane and click OK.*

Once Dictionary Fields is selected the Field Explorer window will open.

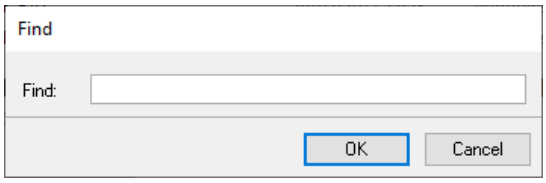



*To insert a Field name, select the resource in the right-hand pane and click OK.*

The following is a description of the Script menu available for the tab:

*Find …*

Use this menu option to open the script editor Find window to search for text. Control-F can be used as a shortcut.

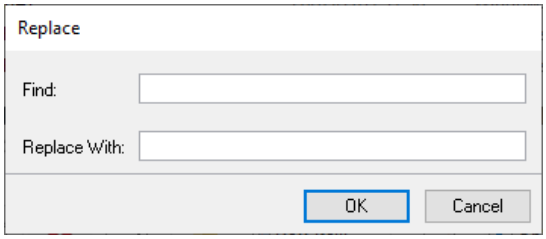



*Find Next*

Use this menu option to find the next occurrence. Control-G can be used as a shortcut.

*Replace …*

Use this menu option to open the script editor Replace window to search and replace text. Control-R can be used as a shortcut.

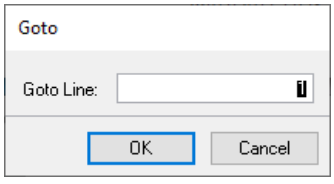



*Replace and Find Next*

Use this menu option to replace and find the next occurrence. Control-B can be used as a shortcut.

*Goto Line …*

Use this menu option to open the script editor Goto Line window to jump to a specified line. Control-N can be used as a shortcut.




*Save and Continue*

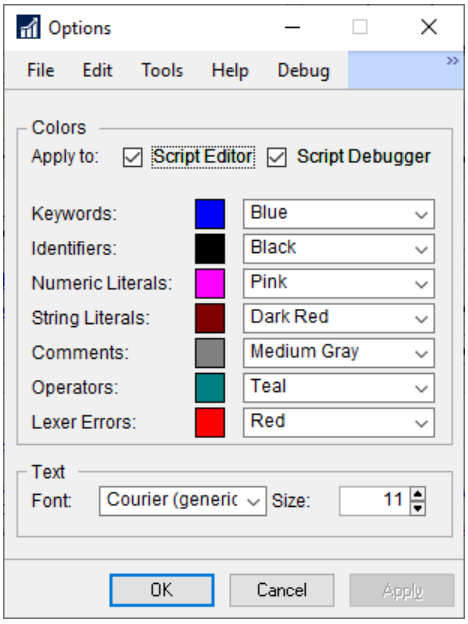
Use this menu option to save the current trigger without clearing the window. Control-S can be used as a shortcut.

*Check Syntax*

Use this menu option to check the syntax of the current script. Any errors will be displayed in a dialog window. Control-K can be used as a shortcut.

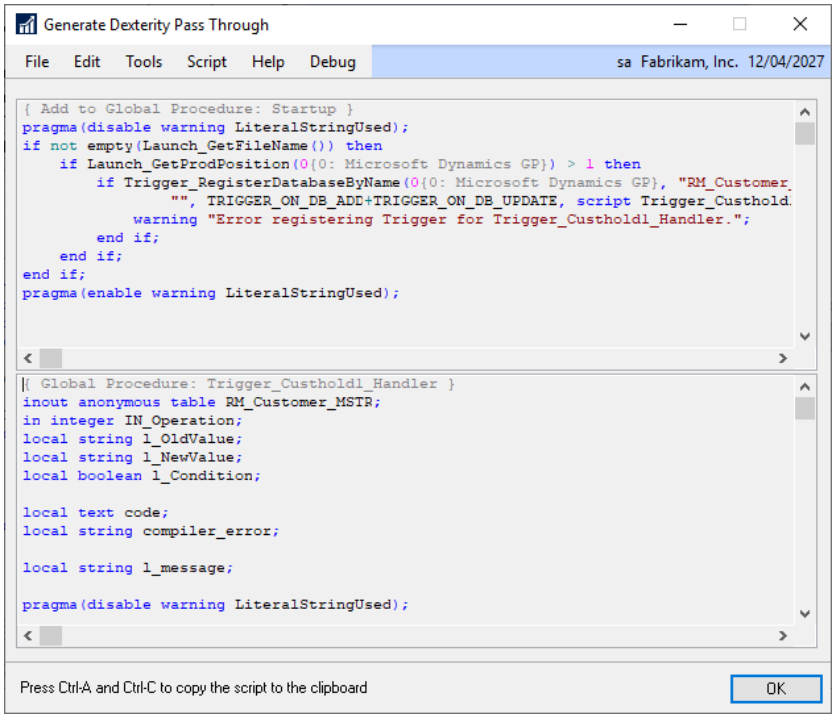
### *Options*

Use this menu option to open the Options window to allow the syntax highlighting colors, font style, and size to be changed. Control-O can be used as a shortcut.




### *Generate Dexterity Pass Through*

Use this menu option to generate Dexterity pass through sanScript code from a trigger script including the trigger registration that can be copied and pasted into a Dexterity development dictionary. Control-D can be used as a shortcut.



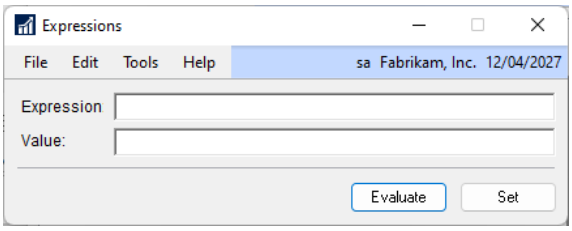
### *Names Button Uses Clipboard*

Use this menu option to control whether the Names Button returns directly to the script (default) or to the clipboard.

*To be able to use the clipboard, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

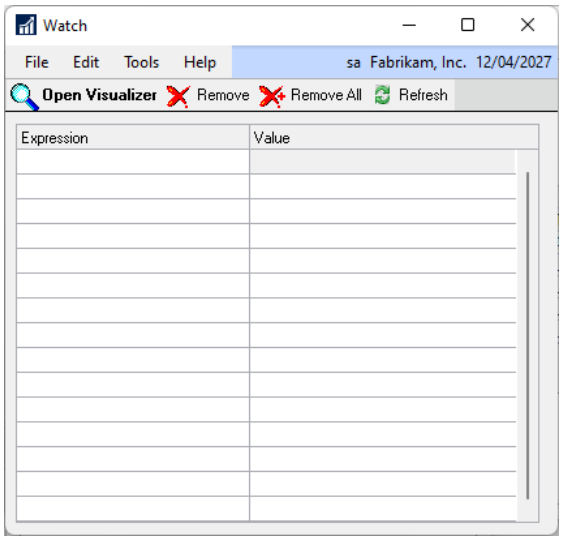
### *Debug Expressions*

Use this menu option to open the Script Debugger Expressions window. When the Expressions window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



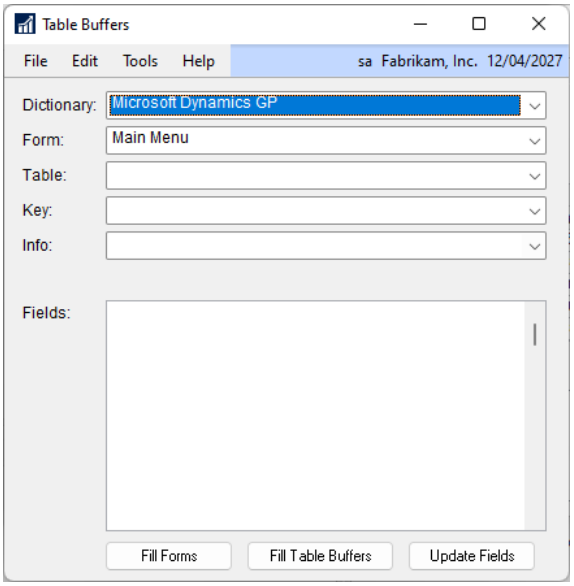

### *Debug Watch*

Use this menu option to open the Script Debugger Watch window. When the Watch window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



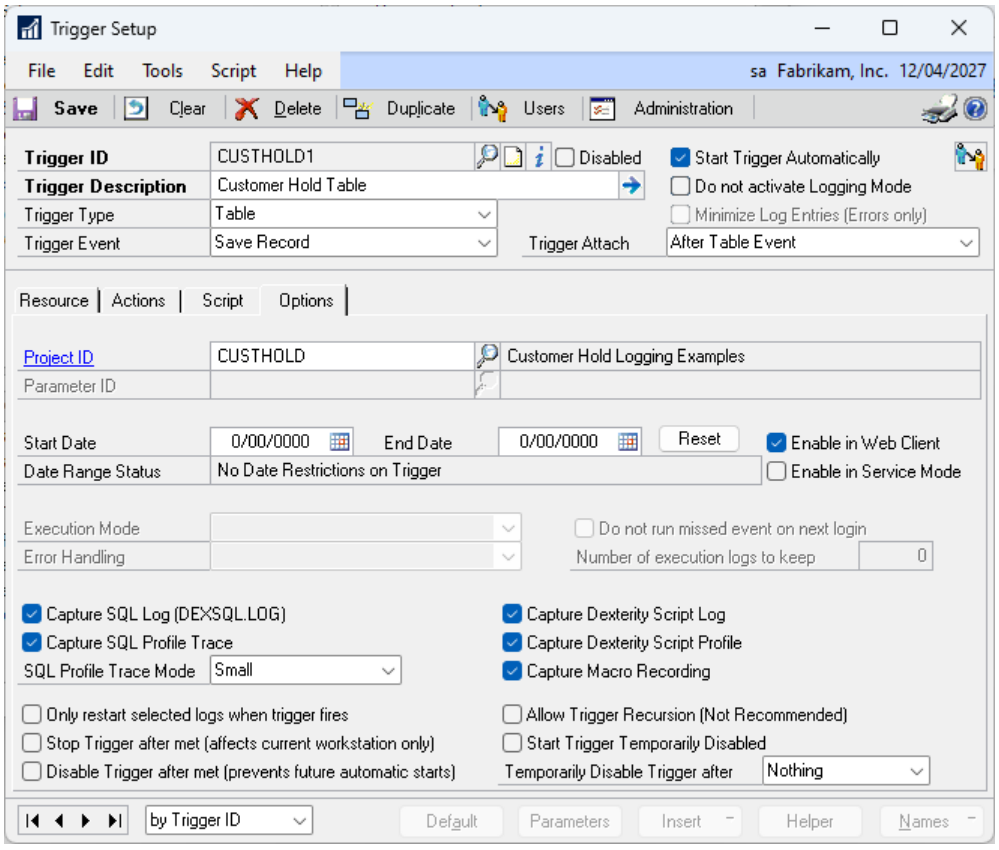
### *Debug Table Buffers*

Use this menu option to open the Script Debugger Table Buffers window.



## Options Tab

The Options tab contains optional settings which change the behavior of the trigger.




The following is a description of the individual script fields on the tab.

*Project ID*

Use this field to add the current trigger to a development project.

*Parameter ID*

For Non Logging Triggers using a Focus Event, Add Form Menu or Add Field Context Menu Type you can specify a Parameter List to be used with the script.

*Start Date*

You can specify a Start Date to restrict the dates that a trigger will automatically start.

*End Date*

You can specify an End Date to restrict the dates that a trigger will automatically start.

*If the Start Date and the End Date are the same, the trigger will only be active for a single day. If the End Date is before the Start Date, then the trigger will be inactive during the date range. The status field will show the behavior based on the selected dates. If a Start Date is not specified, the trigger will be active up to the End Date. If an End Date is not specified, the trigger will be active from the Start Date.*

### *Enable in Web Client*

This checkbox tells GP Power Tools that this trigger should be enabled for the Microsoft Dynamics GP Web Client .

*When importing Triggers that have been exported from a build earlier than Build 31, you will need to edit the trigger and select this option manually if you want the trigger active in the Microsoft Dynamics GP Web Client.*

### *Enable in Service Mode*

This checkbox tells GP Power Tools that this trigger should be enabled when Microsoft Dynamics GP is being executed in Service Mode (for Service Based Architecture) or when in Dynamics Process Server (DPS) mode.

### *Execution Mode*

For Scheduled Event Trigger Type only: Use this drop-down list to select how often the scheduled event should execute. Select from every time, once per system, once per user, once per company or once per user/company combination.

### *Do not run missed event on next login*

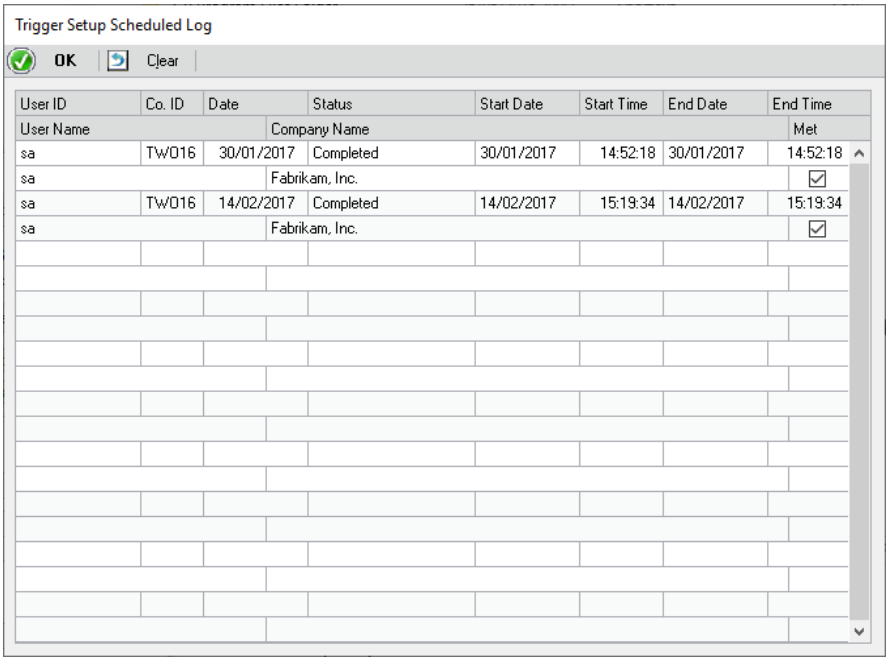
For Scheduled Event Trigger Type only: By default, if a scheduled event is missed because Microsoft Dynamics GP was not logged in at the time it was scheduled, it will execute on the next login. Select this checkbox to skip the missed event and just wait for the next scheduled time.

### *Error Handling*

For Scheduled Event Trigger Type only: Use this drop-down list to select how error are handled. You can select not to retry, to retry once or up to 5 times when the trigger is incomplete (usually caused by a script error), or retry once or up to 5 times when the trigger completed but the conditional script returned false.

### *Number of execution logs to keep*

For Scheduled Event Trigger Type only: The system keeps logs each time a scheduled event is triggered, use this field to specify how much history should be kept. Drilling down on the field will open the Trigger Setup Scheduled Log window:




### *Capture SQL Log*

You can select which of the logging modes to enable, this option enables the SQL Logging when this trigger is active. This option is not valid for Non-logging triggers.

### *Capture SQL Profile Trace*

You can select which of the logging modes to enable, this option enables the SQL Profile Tracing when this trigger is active. This option is not valid for Non-logging triggers.

*SQL Profile Tracing is not enabled until it has been setup using the SQL Profile Trace Settings window under the Administrator Settings.*

### *SQL Profile Trace Mode*

When using SQL Profile Tracing, you can use this option to select the type of SQL Profile Trace created. You can select between Small, Medium, Large and Performance. The Other mode can be used in conjunction with a customized MBS_SQL_Tracing_API_5 stored procedure in the DYNAMICS database. This option is not valid for Non-logging triggers.

### *Capture Dexterity Script Log*

You can select which of the logging modes to enable, this option enables the Dexterity Script Logging when this trigger is active. This option is not valid for Non-logging triggers.

### *Capture Dexterity Script Profile*

You can select which of the logging modes to enable, this option enables the Dexterity Script Profiling when this trigger is active. This option is not valid for Non-logging triggers.

### *Capture Macro Recording*

You can select which of the logging modes to enable, this option enables the Macro Recording when this trigger is active. This option is not valid for Non-logging triggers.

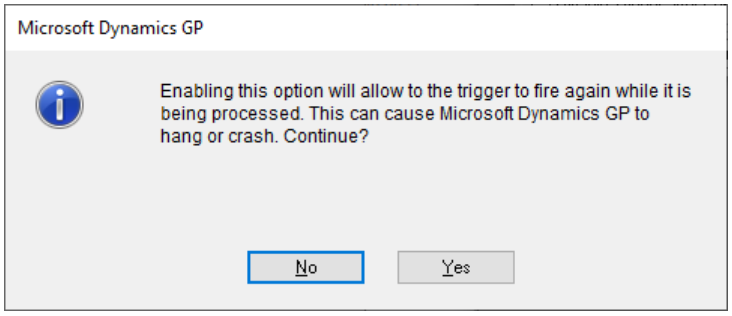
*Macro Recording can only work when a single instance of Microsoft Dynamics GP is running on a workstation, or if multiple instances are running, Macro Recording will only work on the first instance launched.*

### *Only restart selected logs when trigger fires*

Using this checkbox, you can control which logging modes are restarted when the trigger fires. By default, all active logging modes are restarted each time a trigger fires. If this checkbox is enabled, only the logging modes selected for this trigger will be restarted when this trigger fires. This option is not valid for Non-logging triggers.

### *Allow Trigger Recursion*

This checkbox is used to control whether trigger recursion is allowed. It is disabled by default as it can cause infinite loops or crashing. If you try to enable recursion you will receive the following warning. Trigger Recursion can occur when the trigger script performs an action which can fire the same trigger again. For example: a table save trigger, making a change to a table and saving the record again.




### *Stop Trigger after Condition met*

Using this checkbox, you can specify that a trigger should only be used once per session. When the Trigger fires and the condition is met, the trigger will be stopped until next login or manual restart.

### *Disable trigger after Condition met*

Using this checkbox, you can specify that a trigger should only be used once. When the trigger fires and the condition is met, the trigger will be disabled preventing it from starting until it is re-enabled.

### *Start Trigger Temporarily Disabled*

Using this checkbox, you can specify that a trigger should be temporarily disabled immediately after it is registered. Use the MBS_Trigger_EnableSingle and the MBS_Trigger_DisableSingle Helper Function to enable and disable triggers when desired.

### *Temporarily Disable Trigger after*

Use this drop-down list to temporarily disable a trigger again automatically after it has been executed.

## Restriction of Scope

GP Power Tools has a restriction which must be taken into account when using the Automatic Trigger Mode.

When using a table trigger type, GP Power Tools uses a Dexterity database trigger. A Dexterity database trigger is only capable of tracking changes made to the tables using Dexterity commands.

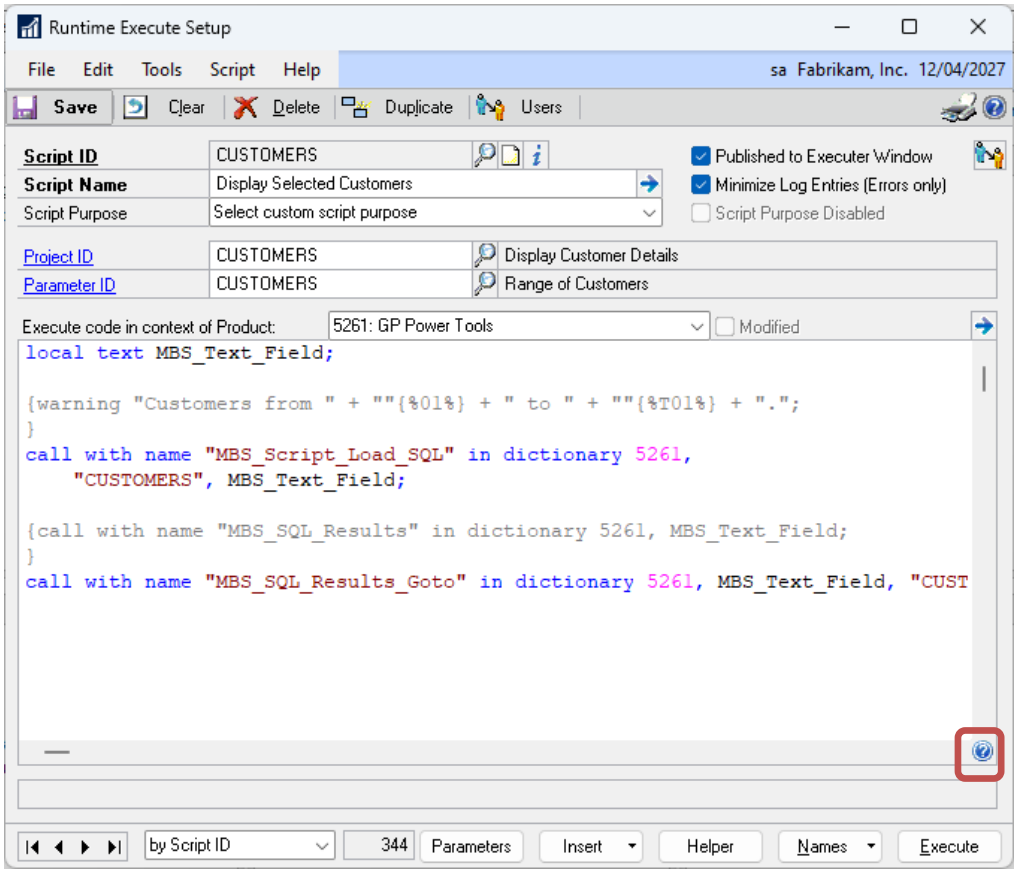
*Changes made to tables using tools or applications other than Dexterity will not be picked up by GP Power Tools. This can include table changes made directly by SQL Query Analyzer, pass-through SQL commands, SQL stored procedures, SQL triggers, or updates from eConnect, Integration Manager's SQL Optimized or Microsoft Dynamics GP eConnect adapters, ADO (ActiveX Data Objects) from VBA (Visual Basic for Applications) or any other external application.*

## Runtime Execute Setup

You can open the Runtime Execute Setup window by selecting Runtime Execute Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Runtime Execute Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The Runtime Execute Setup window can be used to run any Dexterity sanScript code without requiring the Dexterity development environment. Scripts written in this window can be used to manipulate tables using Dexterity commands or to call existing functions and procedures in any dictionary.

Script IDs created in this window can be loaded and executed from an Trigger Setup trigger, another Runtime Execute Setup script or a .Net Execute Setup script. This allows code re-use in a similar fashion to having multiple procedure calls as well as mixing of languages.

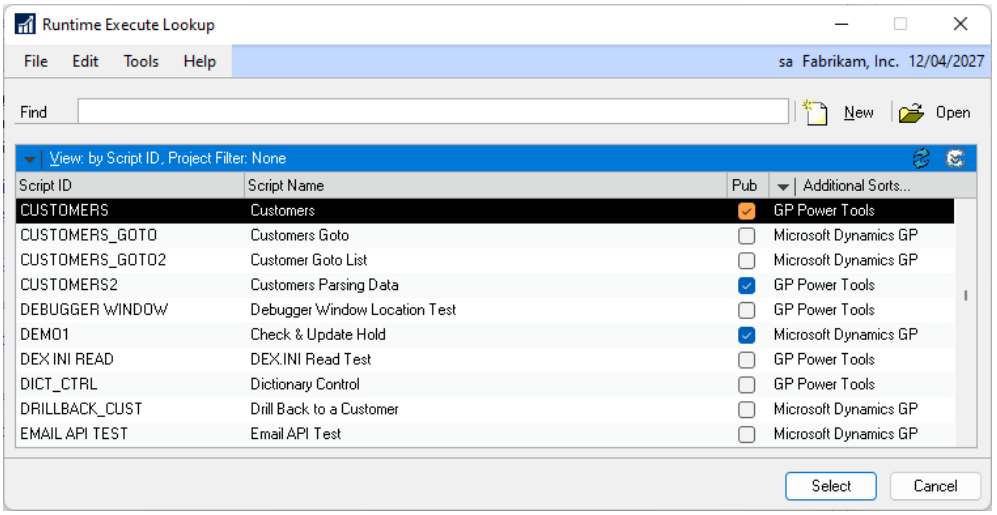



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

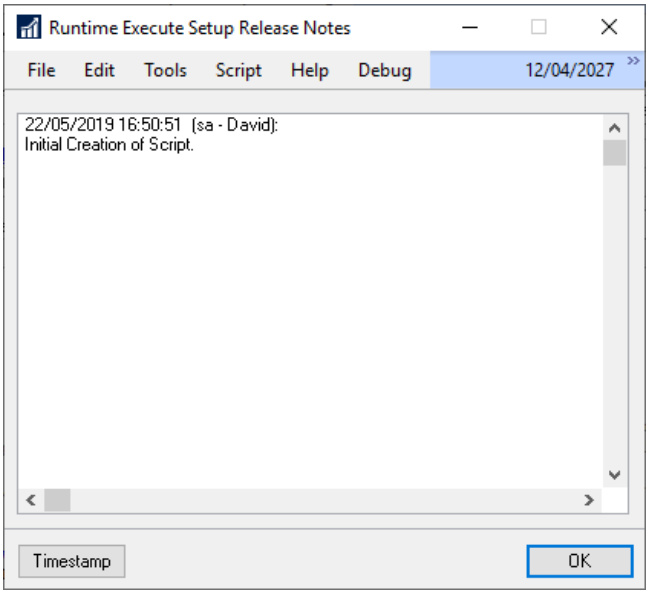
The following is a description of the individual fields on the window:

*Script ID*

This field contains a unique identifier for each Runtime Execute Setup script in the system. The lookup button can be clicked to select from existing script IDs. The lookup will be filtered to the current project if the script belongs to a project.



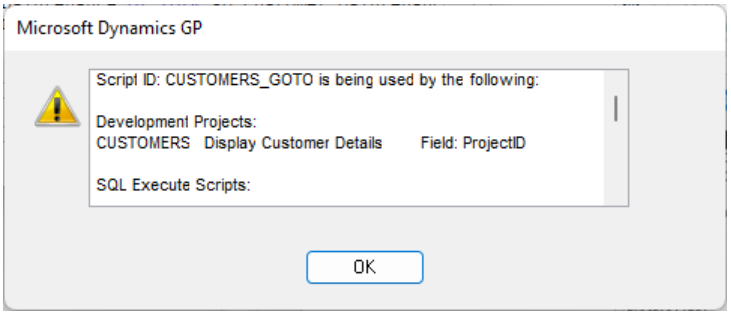

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Script IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*Runtime Execute Information*

Click this button to display what resources are using this script.

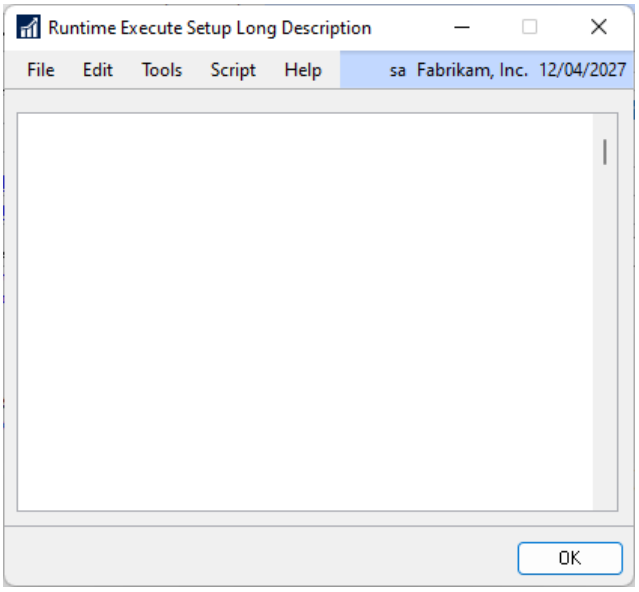



*Script Name*

This field contains a description of the script.

*Long Description*

Use the Script Name expansion button to open the Long Description window. Use this field for a more detailed description of the script. The Long Description is displayed on the Runtime Executer window.




*Script Purpose*

This drop-down list can be used to specify a custom script purpose for the script. Changing the script purpose will replace the script with the template code needed. Purposes include using the script for Report Writer functions (as described in chapter 7), using the script for Service Enabled Procedures, using the script to register custom SmartList Builder Gotos, using the script for handling SQL Gotos, using the script for handling URL Drill Backs, and using the scripts for registering and handling of Visual Studio Integration Toolkit Custom Forms.

*URL Drill Backs allow external applications to all custom scripts in Microsoft Dynamics GP. They are called with a URL in the format below:*

*dgpp://dgpb/?Db=<Instance>&Srv=<Server>&Cmp=<DBName>&Prod=5261&Act=SCRIPT&Func=RuntimeExecute&ID=<ScriptID>&Param=<ParameterList>.*

*Published to Executer Window*

This checkbox indicates if the current script can be accessed from the read only Runtime Executer window.

*Minimize Log Entries*

This option can be enabled to prevent the script generating entries in the GPPTools_<User>_<Company>.log file unless an error occurs.

*Script Purpose Disabled*

This option can be used to disable the script when a Script Purpose is in use.

*Project ID*

Use this field to add the current script to a development project.

*Parameter ID*

Use this field to specify a Parameter List to be used with the script.

*Execute Dexterity SanScript code in the context of Product*

This drop-down list contains a list of products currently installed on the Microsoft Dynamics GP workstation.

*Modified*

This checkbox can be used to force the script to execute in the context of the modified dictionary. This allows the script to reference Modifier added local fields.

*To be able to execute scripts against modified dictionaries, the WinthropDC.GpPowerToolsVC.dll Addins must be installed.*

*Clipboard Button*

This expansion button opens the Runtime Execute Script Clipboard window. This window can be used as temporary storage when editing scripts to allow for cutting and pasting between scripts. Use the Copy Script Button if you wish to copy the current script to the clipboard window and the Clear Script Button to clear the clipboard window contents.


*When changing the script purpose, you will be asked if you want to copy to the clipboard before the script is reset. You can then copy back the portions of the script you want to keep. The clipboard window will close with the Runtime Execute Setup window.*

### *Script*

This text field contains the script to be executed. It cannot have any parameters. The script runs as though it is a global procedure in the context of the dictionary specified in the drop-down list. The script is checked for syntax errors when saved.

*Runtime Execute Setup can be used to manipulate data in tables when complex business logic is required. In this situation writing the equivalent code in Transact SQL can be difficult. You could loop through a range of records in table and conditionally make different changes depending on the data in the records. For example, re-formatting phone numbers in the Customer Master table to different formats depending on whether they are domestic, international or mobile/cell numbers.*

The following is a description of the additional buttons on the window:

*Help Button*

Use this button (highlighted on screenshot) to open the full Dexterity Help file.

*Parameters Button*

Use this button to insert a Parameter Placeholder into the script for the Parameter List selected with the Parameter ID. See the section under Trigger Setup for more information.

*Insert Button*

Use this button to insert a Dexterity sanScript code construct, Parameter Placeholders, .Net Assemblies or code Snippets. See the section under Trigger Setup for more information.

*Helper Button*

Use this button to open the Insert Helper Function window and insert a helper function into the script. See the section under Trigger Setup for more information.

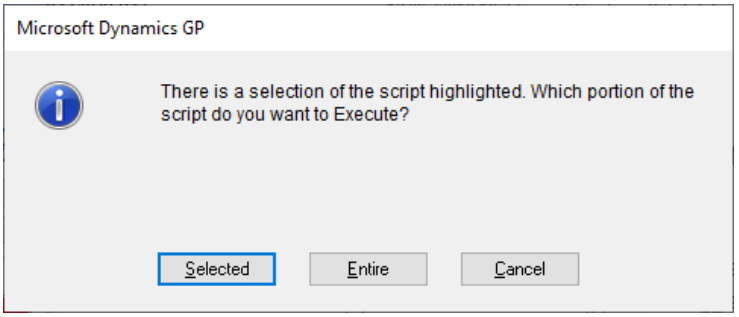
*Names Button*

Use this button to insert a dictionary resource into the script. See the section under Trigger Setup for more information.

*Execute Button*

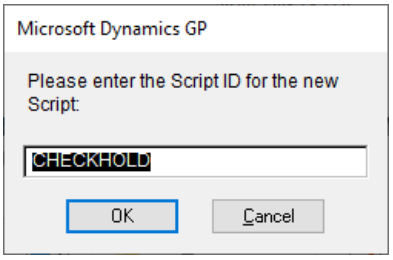
Use this button to execute the script in the context of the dictionary specified. Any compile errors will be shown in the status pane below the script. Execution errors will cause an Exception Error Dialog to open.

*If a selection of the script is currently highlighted, you can decide to execute the highlighted section or the entire script.*



### *Duplicate Button*

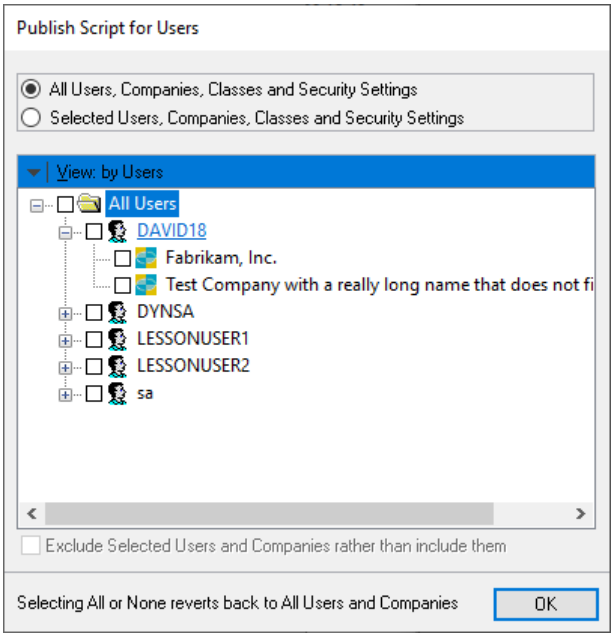
Use this button to duplicate or rename the current script ID to a new script ID. This is useful when an existing script ID is very similar to the new one you want to create.




A new script ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies the script should be published to. Once clicked Publish Script for Users window will open.




You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

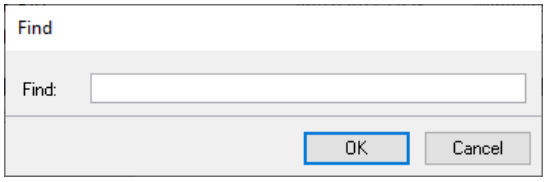
*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the script should not be published to.

The following is a description of the Script menu available for the window:

*Find …*

Use this menu option to open the script editor Find window to search for text. Control-F can be used as a shortcut.

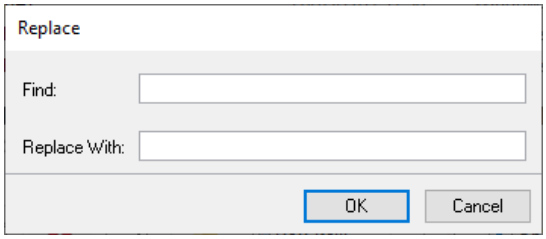



*Find Next*

Use this menu option to find the next occurrence. Control-G can be used as a shortcut.

*Replace …*

Use this menu option to open the script editor Replace window to search and replace text. Control-R can be used as a shortcut.

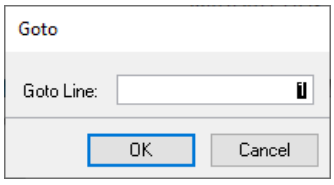



*Replace and Find Next*

Use this menu option to replace and find the next occurrence. Control-B can be used as a shortcut.

*Goto Line …*

Use this menu option to open the script editor Goto Line window to jump to a specified line. Control-N can be used as a shortcut.




*Save and Continue*

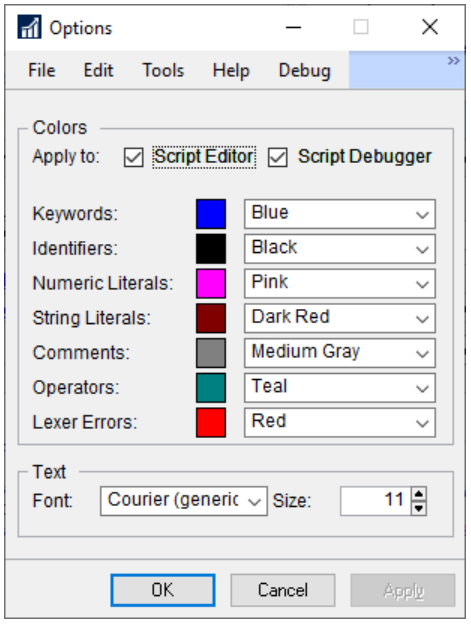
Use this menu option to save the current script without clearing the window. Control-S can be used as a shortcut.

### *Check Syntax*

Use this menu option to check the syntax of the current script. Any errors will be displayed in a dialog window. Control-K can be used as a shortcut.

### *Options*

Use this menu option to open the Options window to allow the syntax highlighting colors, font style, and size to be changed. Control-O can be used as a shortcut.

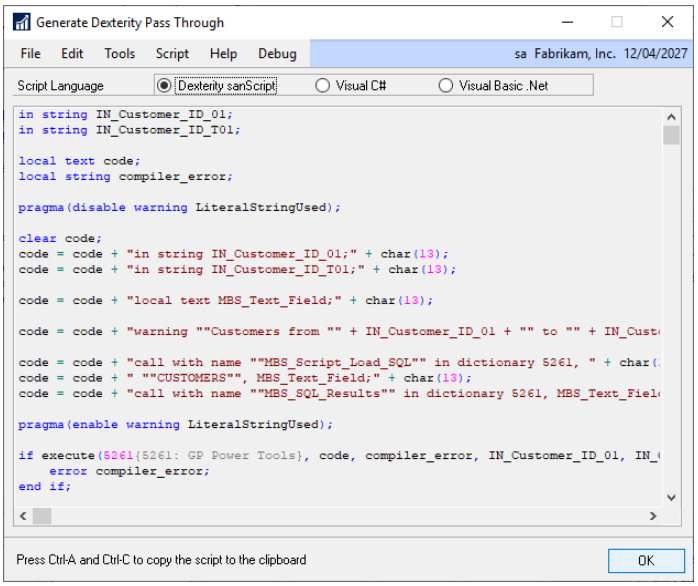



### *Execute*

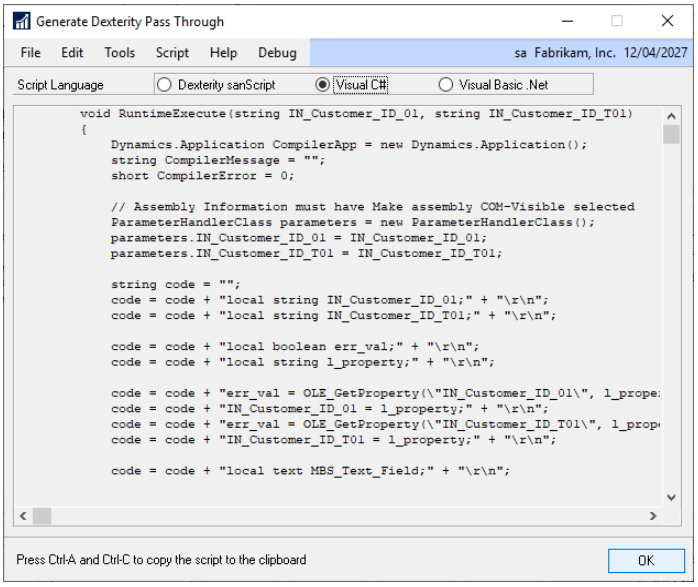
Use this menu option to execute the script. Control-E can be used as a shortcut.

### *Generate Dexterity Pass Through*

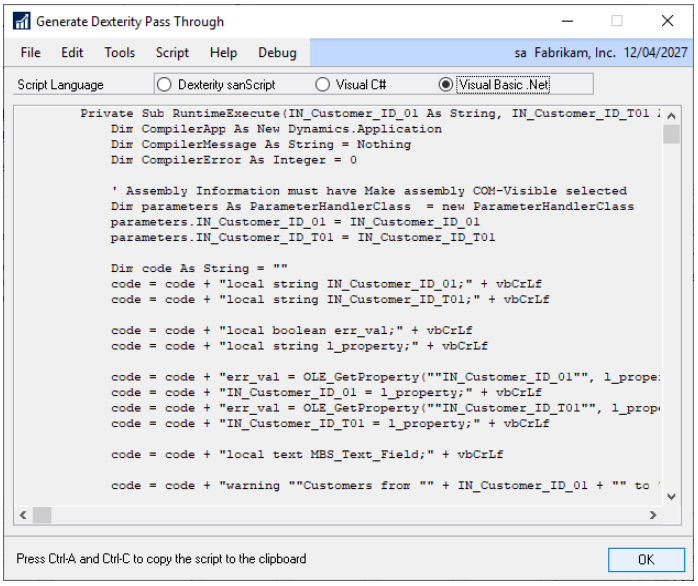
Use this menu option to generate Dexterity pass through sanScript code from a prototype script that can be copied and pasted into a Dexterity development dictionary. Control-D can be used as a shortcut.




Changing the Script Language to Visual C# provides the C# code that can be pasted into a Visual Studio project.




Changing the Script Language to Visual Basic .Net provides the VB code that can be pasted into a Visual Studio project.



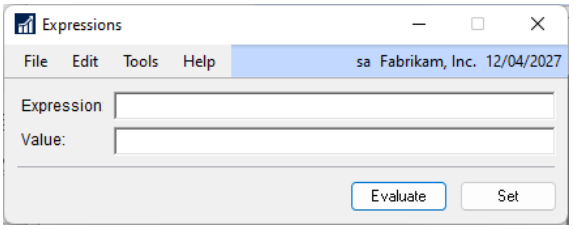

### *Names Button Uses Clipboard*

Use this menu option to control whether the Names Button returns directly to the script (default) or to the clipboard.

*To be able to use the clipboard, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

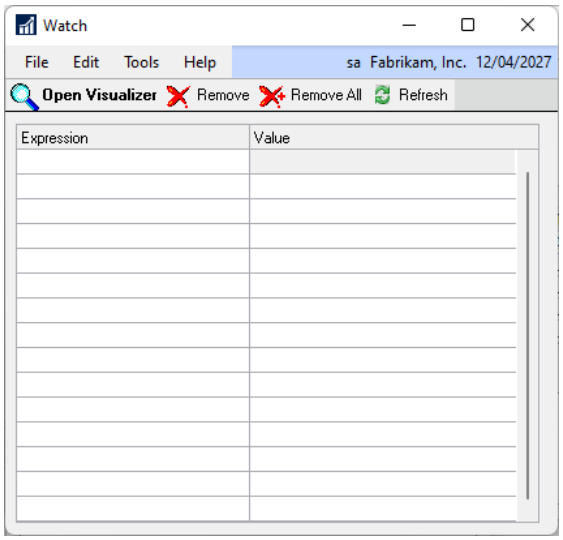
### *Debug Expressions*

Use this menu option to open the Script Debugger Expressions window. When the Expressions window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.



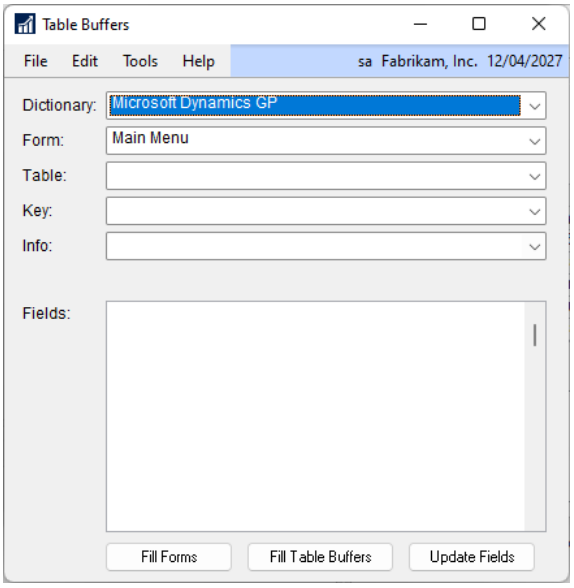
### *Debug Watch*

Use this menu option to open the Script Debugger Watch window. When the Watch window is open, the GP Power Tools Script Debugging Context window will open to allow changing of Dictionary context.




### *Debug Table Buffers*

Use this menu option to open the Script Debugger Table Buffers window.

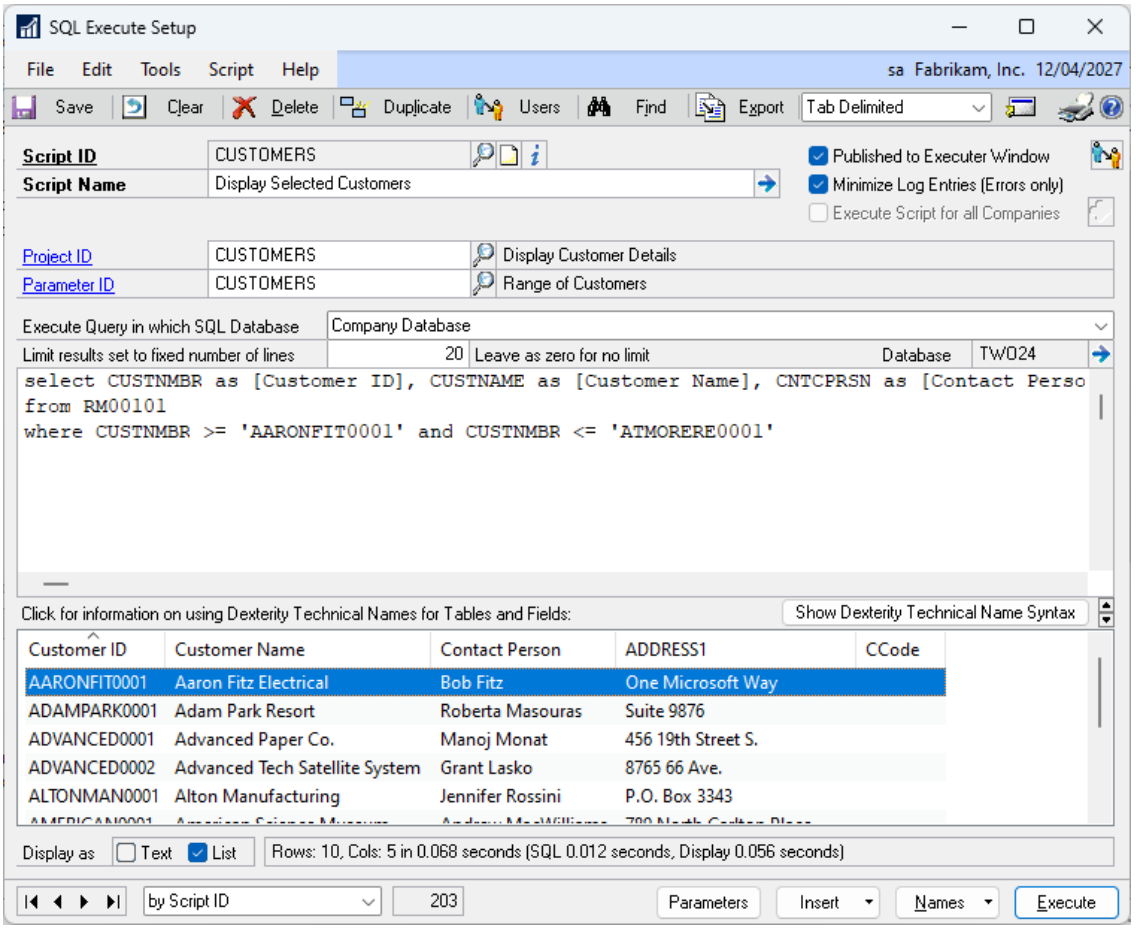


## SQL Execute Setup

You can open the SQL Execute Setup window by selecting SQL Execute Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> SQL Execute Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The SQL Execute Setup window can be used to run any Transact SQL statements without requiring the SQL Administration Tools or MS Query. Commands written in this window can be used to view or manipulate data in any table. This window is like the Query Analyzer window that is installed with the SQL Server client tools.

Script IDs created in this window can be loaded and executed from an Trigger Setup trigger, a Runtime Execute Setup script or a .Net Execute Setup script. This allows a Transact SQL query to be used within Dexterity or .Net code.

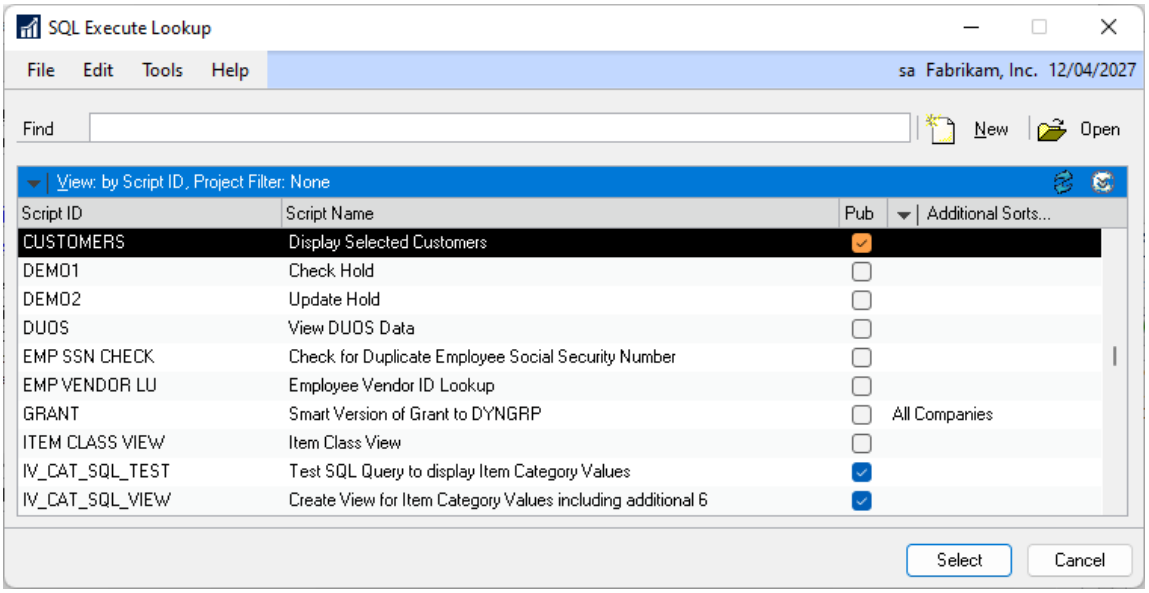



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

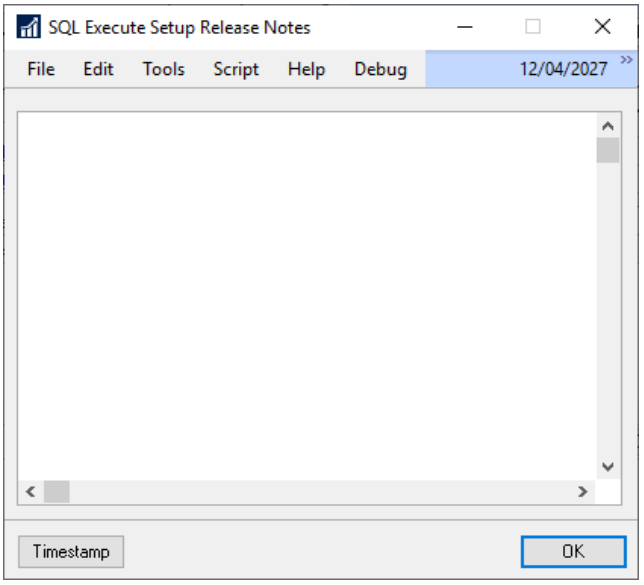
The following is a description of the individual fields on the window:

*Script ID*

This field contains a unique identifier for each SQL Execute Setup script in the system. The lookup button can be clicked to select from existing script IDs. The lookup will be filtered to the current project if the script belongs to a project.



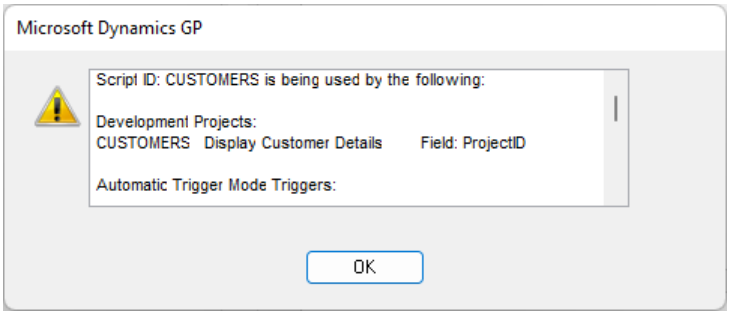

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Script IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*SQL Execute Information*

Click this button to display what resources are using this script.

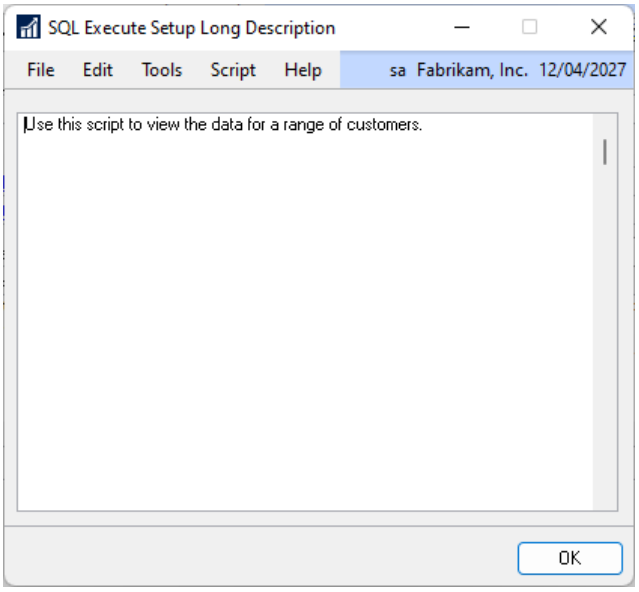



*Script Name*

This field contains a description of the script.

*Long Description*

Use the Script Name expansion button to open the Long Description window. Use this field for a more detailed description of the script. The Long Description is displayed on the SQL Executer window.




*Published to Executer Window*

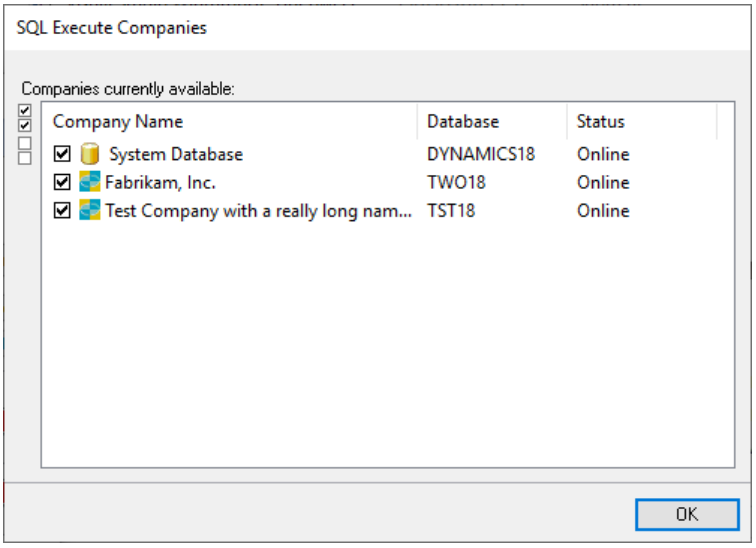
This checkbox indicates if the current script can be accessed from the read only SQL Executer window.

*Minimize Log Entries*

This option can be enabled to prevent the script generating entries in the GPPTools_<User>_<Company>.log file unless an error occurs.

### *Execute Script for all Companies*

A non-published script can be executed against multiple companies using this option. Use the Expansion Button to select companies.




To the left of the company selection list are Mark All and Mark None buttons which can be used to quickly change the selection of the companies. You can also select the System Database if desired. If the selection of the databases is changed from the default (system database not selected and all company databases selected), the selection will be saved along with the script.

### *Project ID*

Use this field to add the current script to a development project.

### *Parameter ID*

Use this field to specify a Parameter List to be used with the script.

### *Execute Query in which SQL Database*

This drop-down list contains a list of SQL databases. The System database and each of the company databases appear in this list.

### *Limit results set to fixed number of lines*

You can use this field to limit the amount of data returned in the results set. Set its value to zero (0) for no limit.

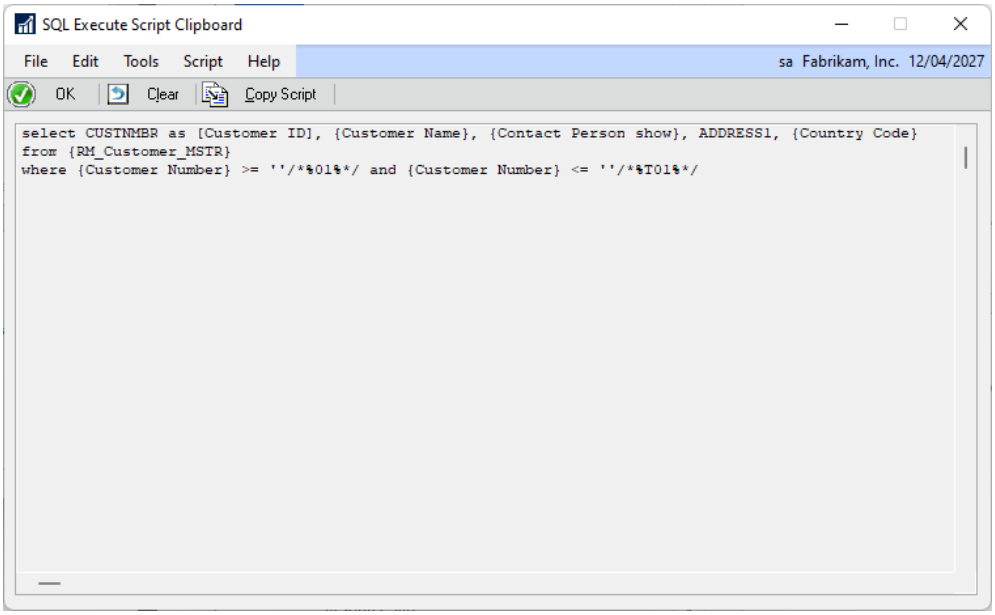
*Setting the value of this field to zero (0) can cause SQL Execute Setup to take a long time to display the results if the returned results set is very large.*

### *Database*

This non-editable field shows the name of the selected SQL Database.

### *Clipboard Button*

This expansion button opens the SQL Execute Script Clipboard window. This window can be used as temporary storage when editing scripts to allow for cutting and pasting between scripts. Use the Copy Script Button if you wish to copy the current script to the clipboard window and the Clear Script Button to clear the clipboard window contents.




*When changing the script purpose, you will be asked if you want to copy to the clipboard before the script is reset. You can then copy back the portions of the script you want to keep. The clipboard window will close with the SQL Execute Setup window.*

### *Script*

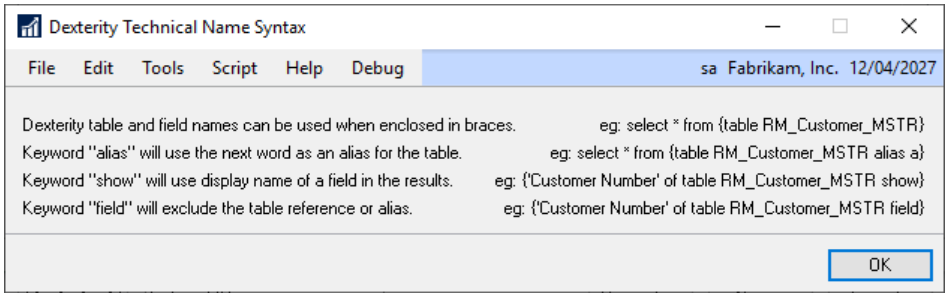
This text field contains the Transact SQL statements to be executed.

*SQL Execute Setup can be used to manipulate data in tables when large set-based changes are required. In this situation writing the equivalent Dexterity sanScript code may not be the most efficient method.*

The following is a description of the additional buttons on the window:

### *Show Dexterity Technical Name Syntax Button*

Use this button to display examples of how Dexterity Technical Names can be used in the script.



### *Divider Adjustment Buttons*

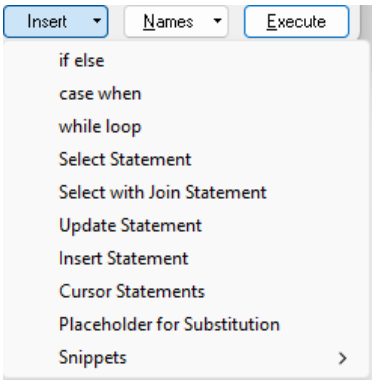
Use these buttons to adjust the position of the horizontal window divider between script and results data.

### *Parameters Button*

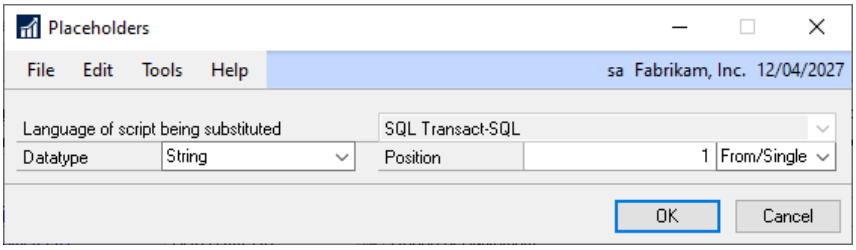
Use this button to insert a Parameter Placeholder into the script for the Parameter List selected with the Parameter ID. See the section under Trigger Setup for more information.

### *Insert Button*

Use this button to insert standard Transact-SQL commands, Parameter Placeholders or code Snippets.




If Placeholder for Substitution is selected, the Placeholders window will open.




If Snippets is selected, the snippets available for the current language are displayed in their groups as specified on the Snippet Setup window.

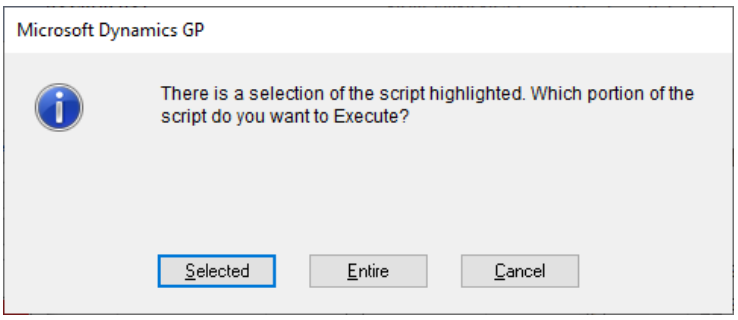
### *Names Button*

Use this button to insert a table, field or constant resource into the script. Once clicked the Table Explorer window will open or if Constants (value) is selected the Constant Explorer window will open. See the section under Trigger Setup for more information.

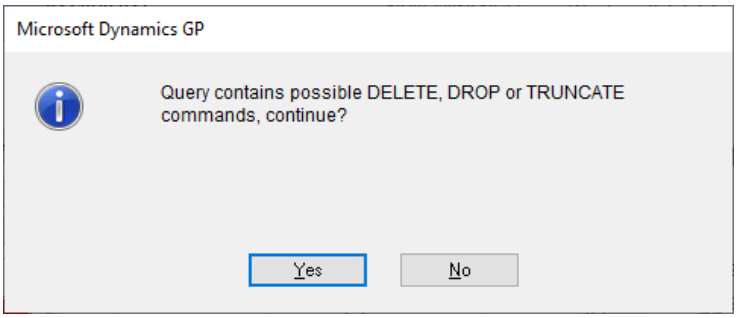
### *Execute Button*

Use this button to execute the script in the context of the SQL database specified. Any execution errors will cause an Exception Error Dialog to open. Results can be shown as unformatted text or as a list.

*If a selection of the script is currently highlighted, you can decide to execute the highlighted section or the entire script.*




*Before the SQL script is executed, it is checked for possible damaging commands and if they exist an additional confirmation is required.*



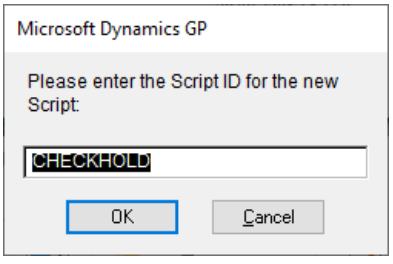

*If a GO Statement is used in the script, make sure it is used at the beginning of the line. When GO Statements exist, the script will be executed in sections with one section for each GO statement. Only the final section will display results, so place any select statements to be displayed after the final GO Statement.*

Dexterity table and field names can be used in the Transact SQL when surrounded by braces { }. They will be converted to the equivalent physical names prior to the code being executed.

The alias keyword can be used to specify an alias other than the table's physical name. The show keyword can be used to display the field's Dexterity display name as the column name. The field keyword is used to limit the generated physical equivalents to be only the column name without the table name or alias prefix.

### *Duplicate Button*

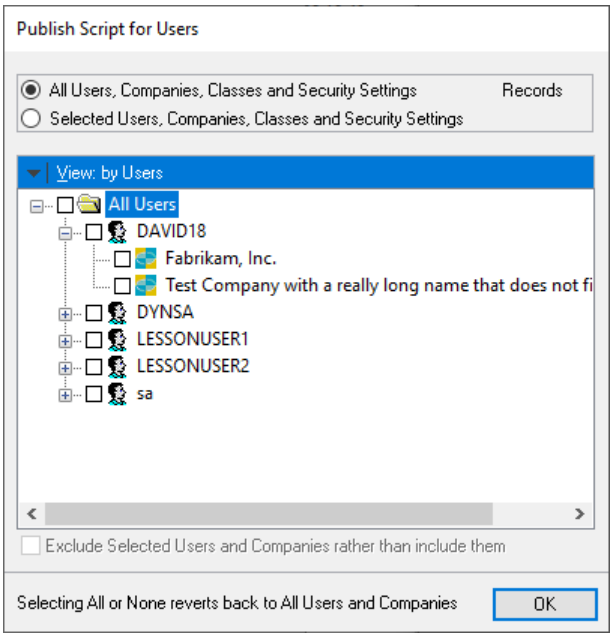
Use this button to duplicate or rename the current script ID to a new script ID. This is useful when an existing script ID is very similar to the new one you want to create.



A new script ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies the script should be published to. Once clicked Publish Script for Users window will open.



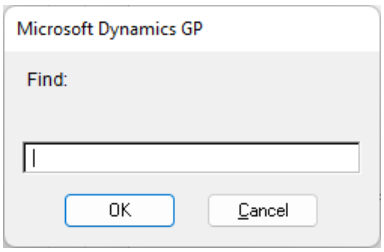

You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the script should not be published to.

### *Find Button*

Use this button to search the results list for the specified search string. You can select to search All Columns or the current Sort Column and whether the search should be a Contains search or a Begins With search.




### *Export Button*

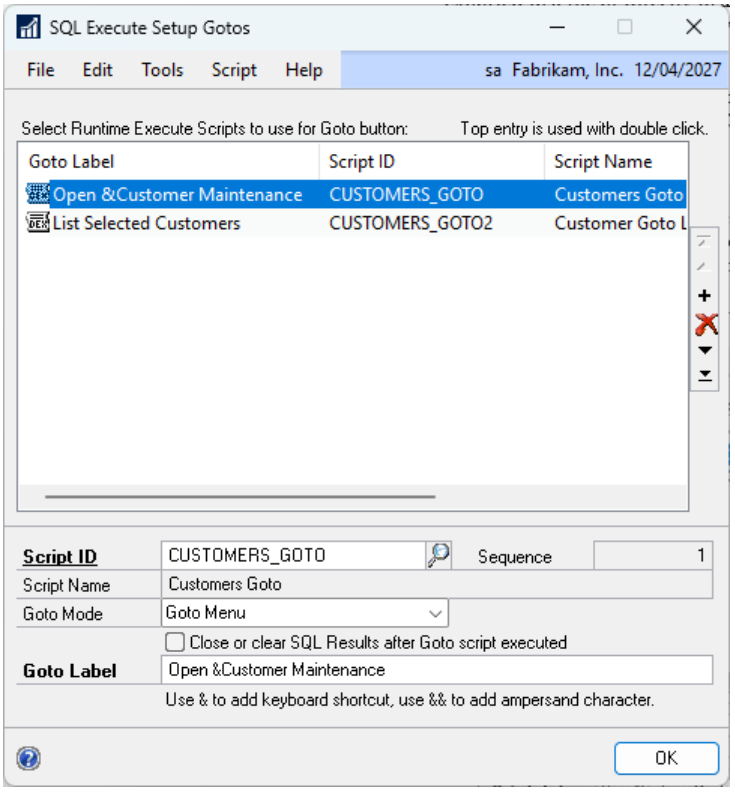
This button will allow the result set displayed in the list view to be exported to a file or directly to an email. The default email settings can be set up in the Email Settings window.

### *Export Mode*

Use this drop-down list to select the format for the exported file. The file can be exported as Tab Delimited, Comma Delimited or as a HTML Table.

### *Gotos Button*

Use this button drop-down menu to setup SQL Gotos or execute an existing SQL Goto on the selected rows in the returned data. You can also right click on the results list.



SQL Gotos allow further actions to be taken on the selected rows of the data returned from SQL Execute scripts. This feature uses Runtime Executer Setup scripts with the Script Purpose set to SQLExecuteGotoHandler to define the required actions and the SQL Execute Setup Gotos window to configure the label to display on the Goto Button and the order of the SQL Gotos. The Runtime Executer Setup Script is executed for each selected line in the result set.

Use the Add Button to add a new SQL Goto and then select the Script ID and define the label to display on the Goto Button. Use "&" if you wish to add a keyboard shortcut and "&&" if you want to add an ampersand. The order of the SQL Gotos to be changed using the Top, Up, Down and Bottom buttons.

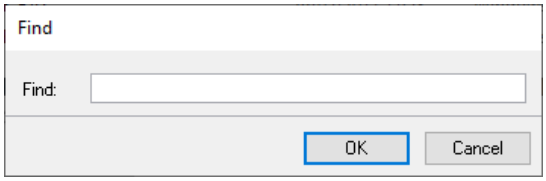
Mark the checkbox if you want to close or clear the SQL Results after Goto script has been executed.

*The Goto Mode drop down list allows scripts to be executed after the query is displayed or before the window is closed. These can be used to set and store the sort column if you want it remembered, or automatically scroll through the displayed data.*

The following is a description of the Script menu available for the window:

*Find …*

Use this menu option to open the script editor Find window to search for text. Control-F can be used as a shortcut.

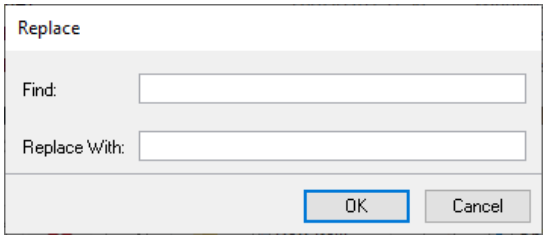



*Find Next*

Use this menu option to find the next occurrence. Control-G can be used as a shortcut.

*Replace …*

Use this menu option to open the script editor Replace window to search and replace text. Control-R can be used as a shortcut.

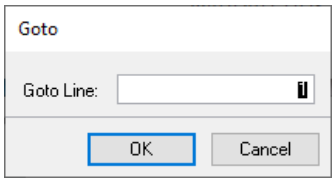


*Replace and Find Next*

Use this menu option to replace and find the next occurrence. Control-B can be used as a shortcut.

*Goto Line …*

Use this menu option to open the script editor Goto Line window to jump to a specified line. Control-N can be used as a shortcut.




*Save and Continue*

Use this menu option to save the current script without clearing the window. Control-S can be used as a shortcut.

*Check Syntax*

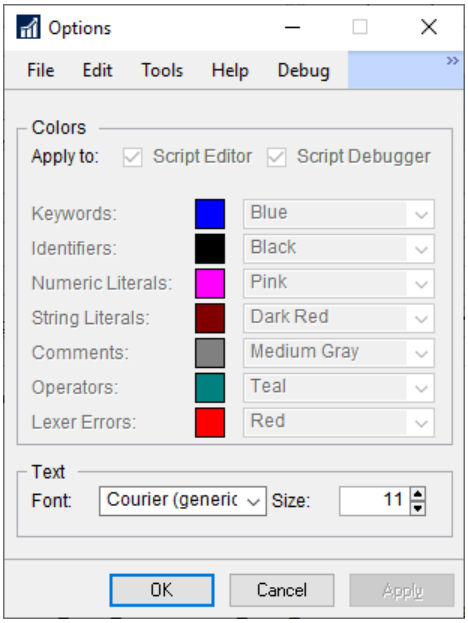
Use this menu option to check the syntax of Dexterity resource names contained in curly braces in the current script. Control-K can be used as a shortcut.

*Convert References*

Use this menu option to convert the Dexterity resource names contained in curly braces in the current script to their SQL equivalents. Control-0 can be used as a shortcut.

*Options*

Use this menu option to open the Options window to allow the font style and size to be changed. Control-O can be used as a shortcut.



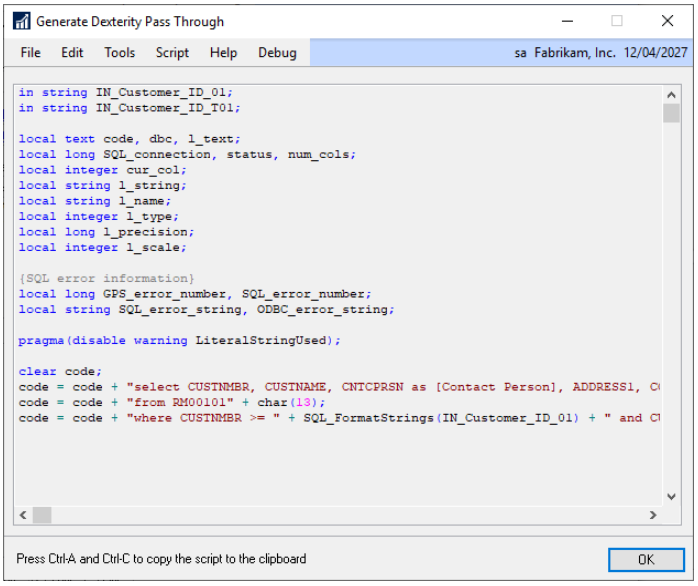

*The Color options are disabled because the syntax highlighting is not available for SQL scripts.*

### *Execute*

Use this menu option to execute the script. Control-E can be used as a shortcut.

### *Generate Dexterity Pass Through*

Use this menu option to generate Dexterity pass through sanScript code from a prototype script that can be copied and pasted into a Dexterity development dictionary. Control-D can be used as a shortcut.




### *Names Button Uses Clipboard*

Use this menu option to control whether the Names Button returns directly to the script (default) or to the clipboard.

*To be able to use the clipboard, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

### *Names Button Uses Fully Qualified Names*

Use this menu option to control whether the Names Button returns SQL table and columns as fully qualified, or just as table and column names.

### *Names Button Adds Keyword 'Show'*

Use this menu option to control whether the Names Button automatically adds the keyword "show" to returned SQL column names so that they will be displayed as Technical Names rather than Physical Names in the SQL Query. It adds the "as" clause to column names.

Following are some examples of using the resource name conversions and keywords:

```
select * from {table RM_Customer_MSTR}
```

is converted to

```
select * from RM00101
```

```
select {'Customer Number' of table RM_Customer_MSTR}
from {table RM_Customer_MSTR}
```

is converted to

```
select RM00101.CUSTNMBR
from RM00101
```

```
select {'Customer Number' of table RM_Customer_MSTR show}
from {table RM_Customer_MSTR}
```

is converted to

```
select RM00101.CUSTNMBR as [Customer Number]
from RM00101
```

```
select {'Customer Number' of table RM_Customer_MSTR field}
from {table RM_Customer_MSTR}
```

is converted to

```
select CUSTNMBR
from RM00101
```

```
select {'Customer Number' of table RM_Customer_MSTR show field}
from {table RM_Customer_MSTR}
```

is converted to

```
select CUSTNMBR as [Customer Number]
from RM00101
```

```
select {'Customer Number' of table RM_Customer_MSTR show alias a}
from {table RM_Customer_MSTR alias a}
```

is converted to

```
select a.CUSTNMBR as [Customer Number]
from RM00101 a
```

The "table" keyword, specifying the table for a field, and surrounding field names containing spaces with single quotes are optional, so

```
select {Customer Number}
from {RM_Customer_MSTR}
```

is converted to

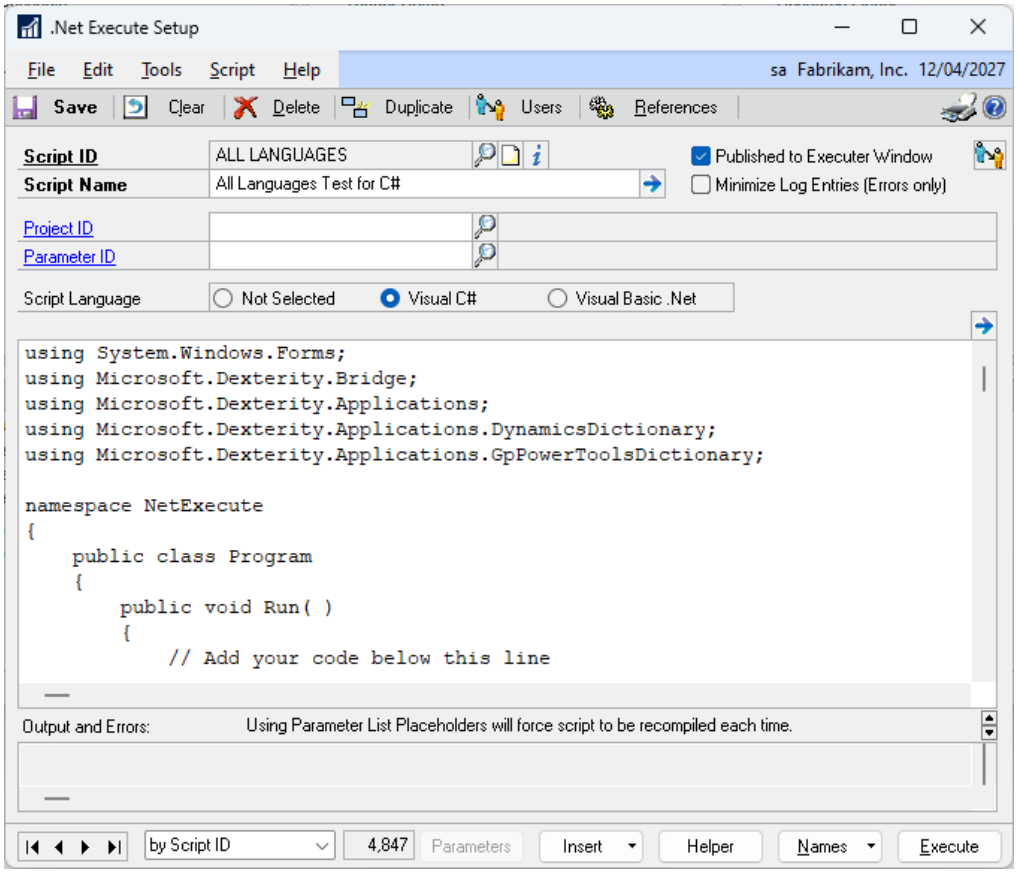
```
select CUSTNMBR
from RM00101
```

## .Net Execute Setup

You can open the .Net Execute Setup window by selecting .Net Execute Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> .Net Execute Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The .Net Execute Setup window can be used to run any Visual C# or Visual Basic.Net code without requiring the Visual Studio development environment. Scripts written in this window can use the form, window, table and field resources from any dictionary or to call existing functions and procedures in any dictionary.

Script IDs created in this window can be loaded and executed from an Trigger Setup trigger, a Runtime Execute Setup script or another .Net Execute script. This allows code re-use in a similar fashion to having multiple procedure calls as well as mixing of languages.



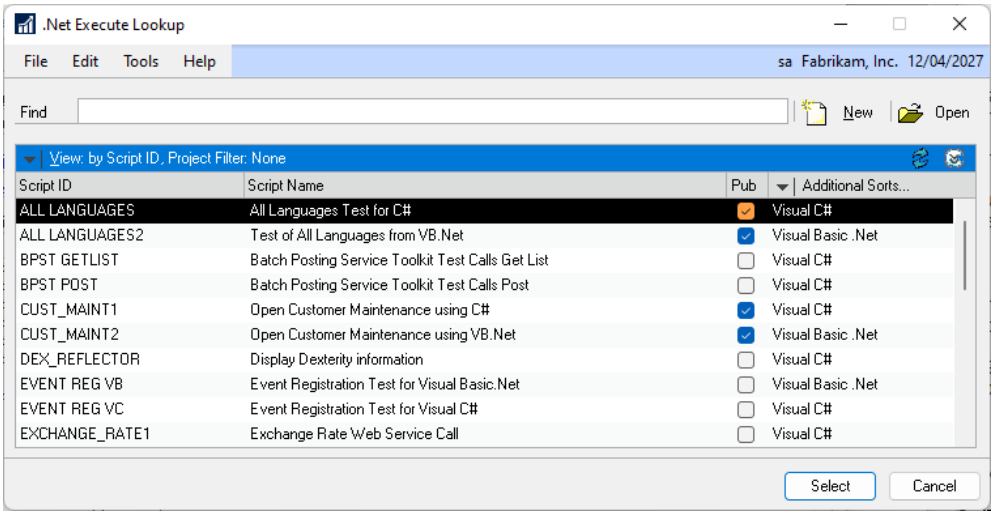

*To be able to execute scripts, the WinthropDC.GpPowerToolsVC.dll and WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

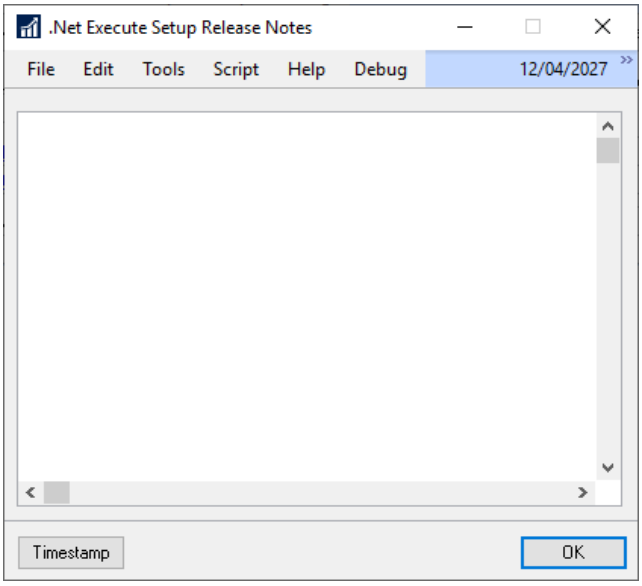
The following is a description of the individual fields on the window:

*Script ID*

This field contains a unique identifier for each .Net Execute Setup script in the system. The lookup button can be clicked to select from existing script IDs. The lookup will be filtered to the current project if the script belongs to a project.

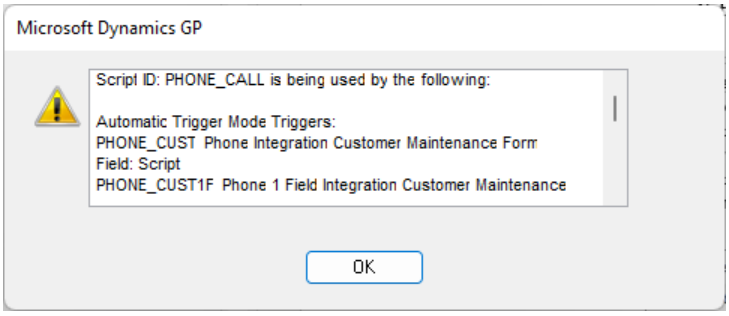



The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*.Net Execute Information*

Click this button to display what resources are using this script.

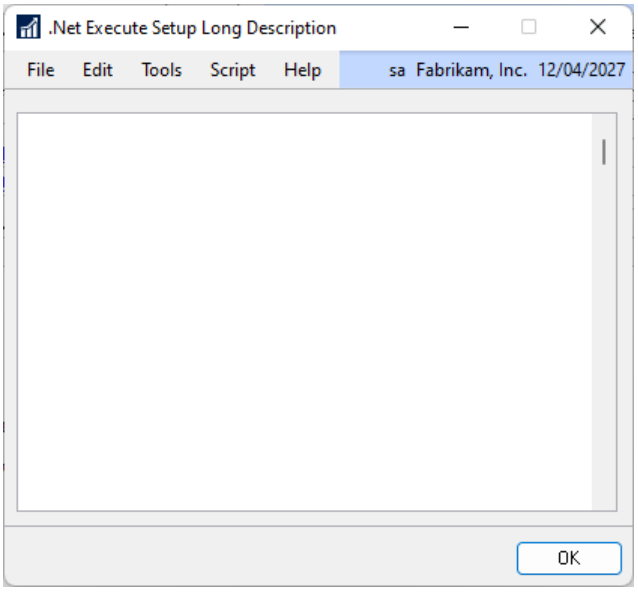



*Script Name*

This field contains a description of the script.

*Long Description*

Use the Script Name expansion button to open the Long Description window. Use this field for a more detailed description of the script. The Long Description is displayed on the .Net Executer window.




*Published to Executer Window*

This checkbox indicates if the current script can be accessed from the read only .Net Executer window.

*Minimize Log Entries*

This option can be enabled to prevent the script generating entries in the GPPTools_<User>_<Company>.log file unless an error occurs.

*Project ID*

Use this field to add the current script to a development project.

*Parameter ID*

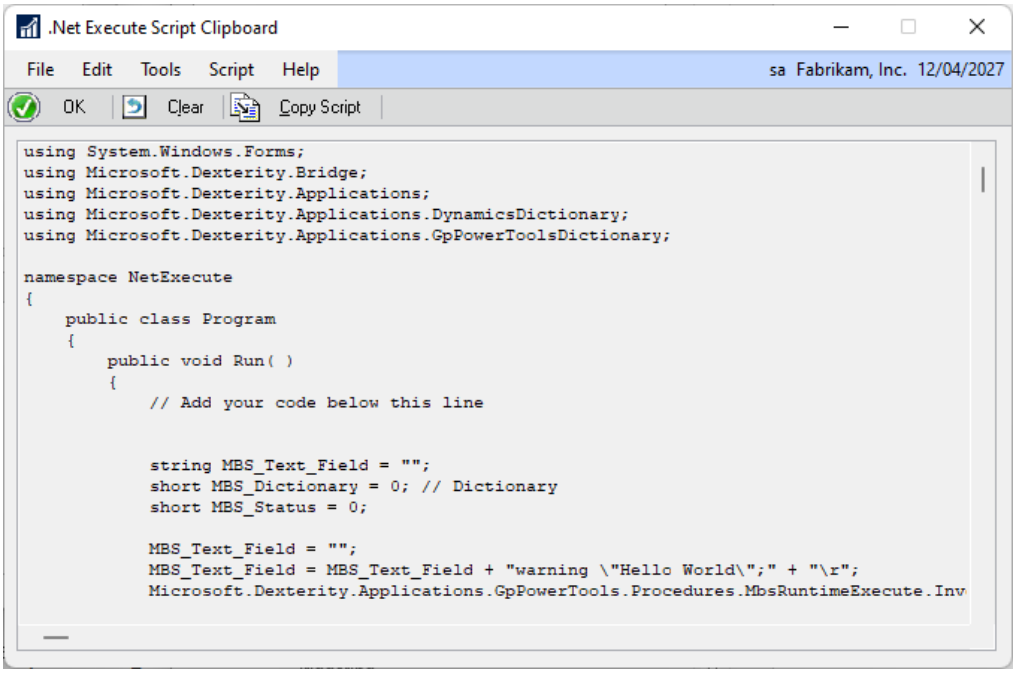
Use this field to specify a Parameter List to be used with the script.

### *Script Language*

This field is used to select the .Net language to be used for this script. You can select from Visual C# or Visual Basic.Net#. When changing the Script Language, the existing script (if any) will be replaced with the base template for the selected language.

### *Clipboard Button*

This expansion button opens the .Net Execute Script Clipboard window. This window can be used as temporary storage when editing scripts to allow for cutting and pasting between scripts. Use the Copy Script Button if you wish to copy the current script to the clipboard window and the Clear Script Button to clear the clipboard window contents.




*When changing the script purpose, you will be asked if you want to copy to the clipboard before the script is reset. You can then copy back the portions of the script you want to keep. The clipboard window will close with the .Net Execute Setup window.*

### *Script*

This text field contains the script to be executed. It cannot have any parameters. The script is checked for syntax errors when saved.

*Changing the name of the Run() function to RunOnce() allows the .Net code to only be execute one time for the current application instance. This can be used to register events with handlers added to the script and only register them once per session.*

The following is a description of the additional buttons on the window:

### *Divider Adjustment Buttons*

Use these buttons to adjust the position of the horizontal window divider between script and output data.

### *Parameters Button*

Use this button to insert a Parameter Placeholder into the script for the Parameter List selected with the Parameter ID. See the section under Trigger Setup for more information.

### *Insert Button*

Use this button to insert a Visual C# or Visual Basic.Net code construct, Parameter Placeholders, .Net Assemblies or code Snippets. See the section under Trigger Setup for more information.

### *Helper Button*

Use this button to open the Insert Helper Function window and insert a helper function into the script. See the section under Trigger Setup for more information.

### *Names Button*

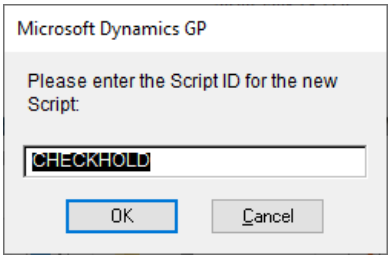
Use this button to insert a dictionary resource into the script. See the section under Trigger Setup for more information.

### *Execute Button*

Use this button to execute the script in the context of the dictionary specified. Any compile errors will be shown in the status pane below the script.

### *Duplicate Button*

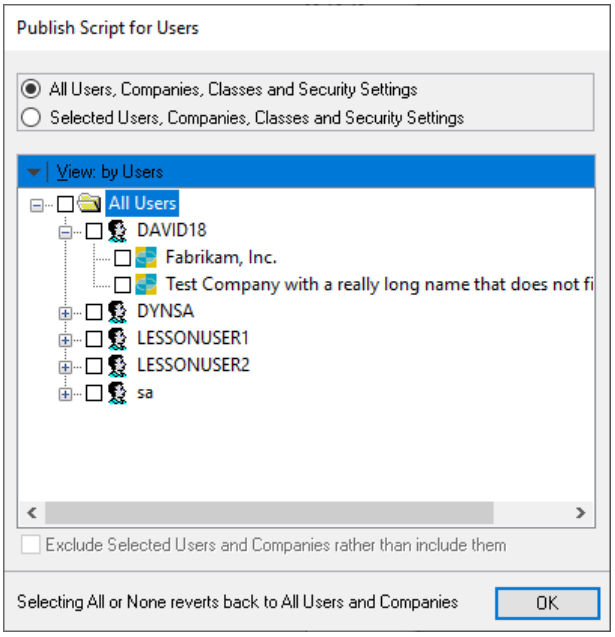
Use this button to duplicate or rename the current script ID to a new script ID. This is useful when an existing script ID is very similar to the new one you want to create.




A new script ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies the script should be published to. Once clicked Publish Script for Users window will open.



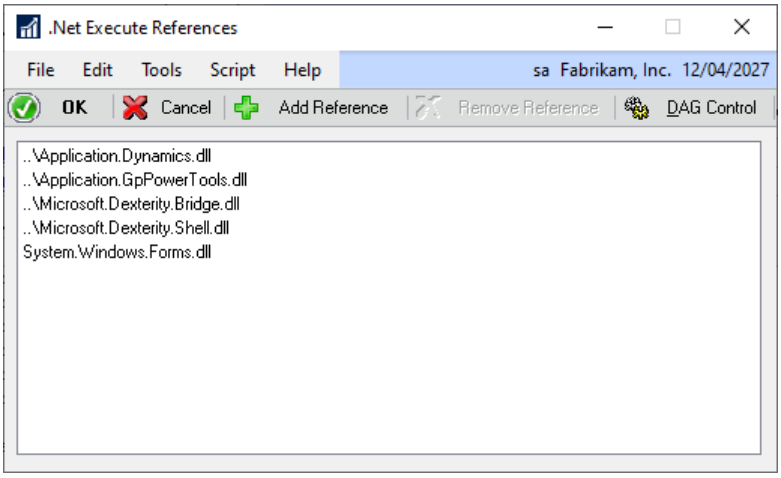

You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the script should not be published to.

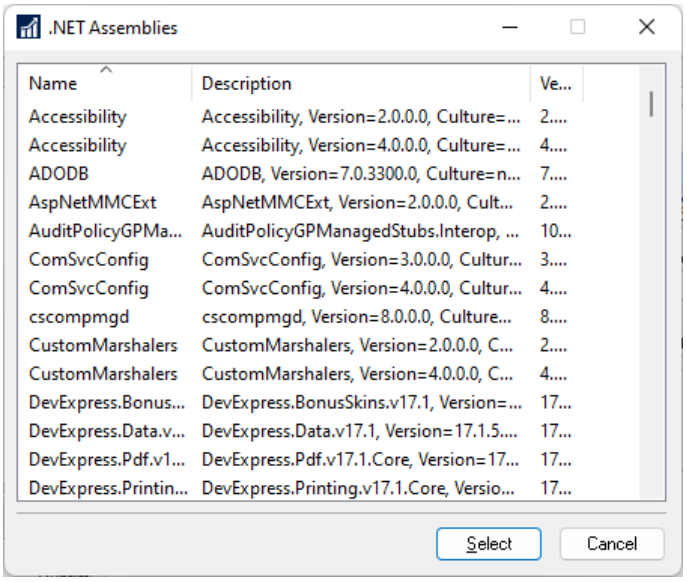
### *References Button*

Use this open the .Net Execute References window. This window can be used to add additional References to a .Net Assembly, dictionary assembly or system DLL (Dynamic Link Library).




*If the Dictionary Assembly for a product dictionary is not available, click on the DAG Control Button to open the Dictionary Assembly Generator Control window which can be used to generate it.*

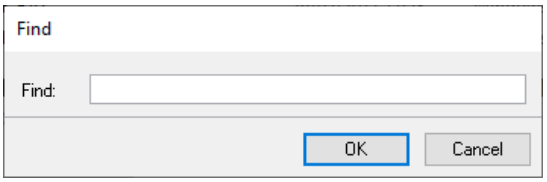
If .Net Assembly is selected, the .Net Assemblies window will open.



The following is a description of the Script menu available for the window:

*Find …*

Use this menu option to open the script editor Find window to search for text. Control-F can be used as a shortcut.

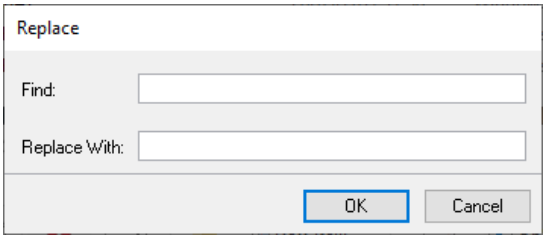



*Find Next*

Use this menu option to find the next occurrence. Control-G can be used as a shortcut.

*Replace …*

Use this menu option to open the script editor Replace window to search and replace text. Control-R can be used as a shortcut.

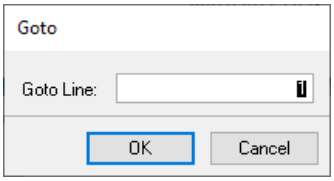



*Replace and Find Next*

Use this menu option to replace and find the next occurrence. Control-B can be used as a shortcut.

*Goto Line …*

Use this menu option to open the script editor Goto Line window to jump to a specified line. Control-N can be used as a shortcut.




*Save and Continue*

Use this menu option to save the current script without clearing the window. Control-S can be used as a shortcut.

*Check Syntax*

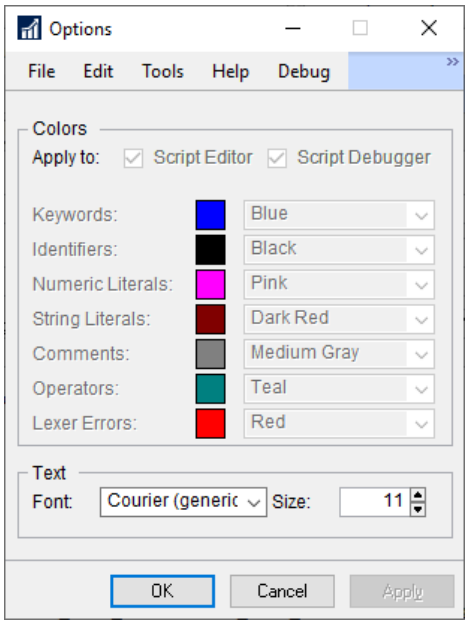
Use this menu option to check the syntax of the current script. Any errors will be displayed in a dialog window. Control-K can be used as a shortcut.

### *References*

Use this menu option to open the .Net Execute References window. Control-0 can be used as a shortcut.

### *Options*

Use this menu option to open the Options window to allow the font style, and size to be changed. Control-O can be used as a shortcut.




*The Color options are disabled because the syntax highlighting is not available for .Net scripts.*

### *Execute*

Use this menu option to execute the script. Control-E can be used as a shortcut.

### *Names Button Uses Clipboard*

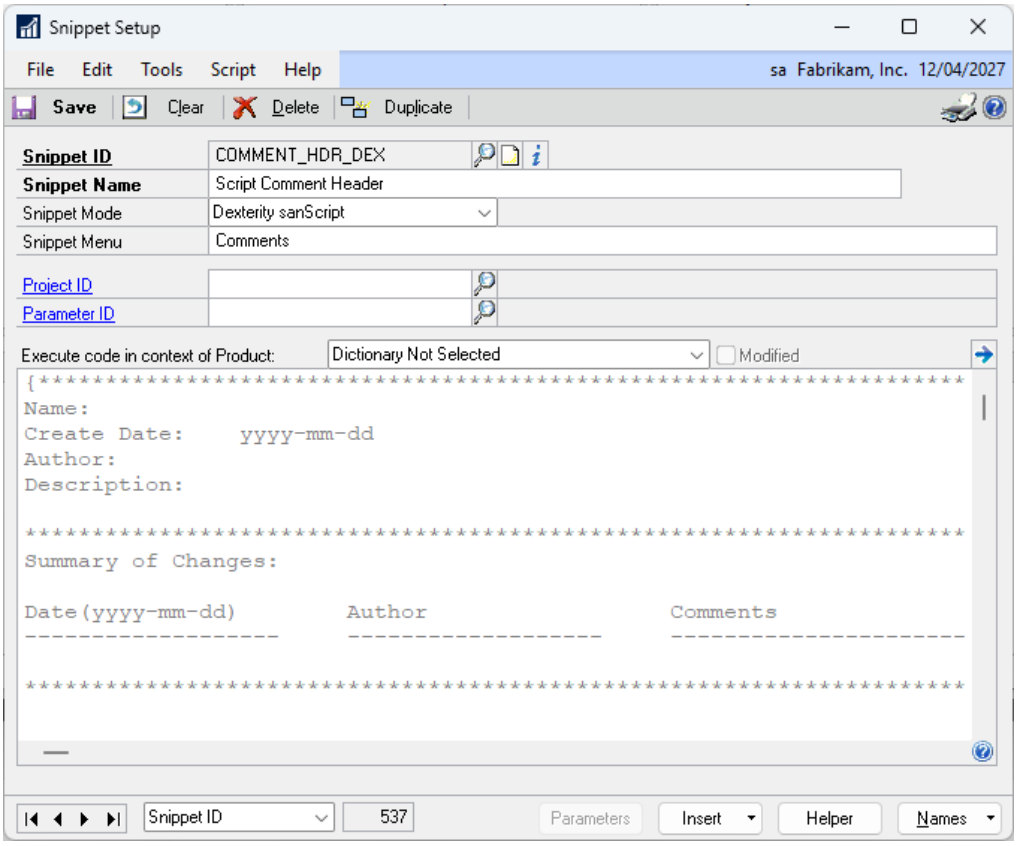
Use this menu option to control whether the Names Button returns directly to the script (default) or to the clipboard.

*To be able to use the clipboard, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

## Snippet Setup

You can open the Snippet Setup window by selecting Snippet Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Snippet Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

The Snippet Setup window can be used to create and store snippets of code that can be inserted into the script editors for the Trigger Setup, Runtime Execute Setup, SQL Execute Setup, and .Net Execute Setup windows.

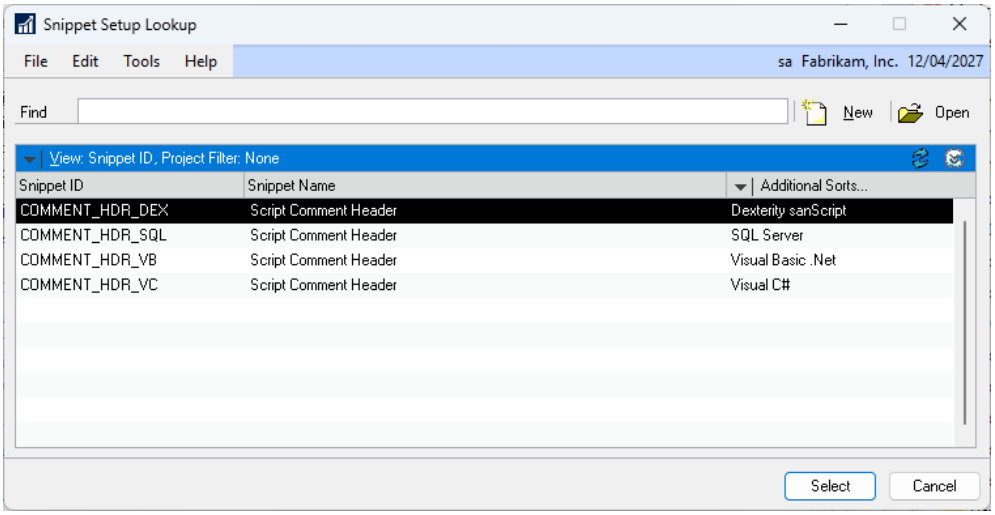



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

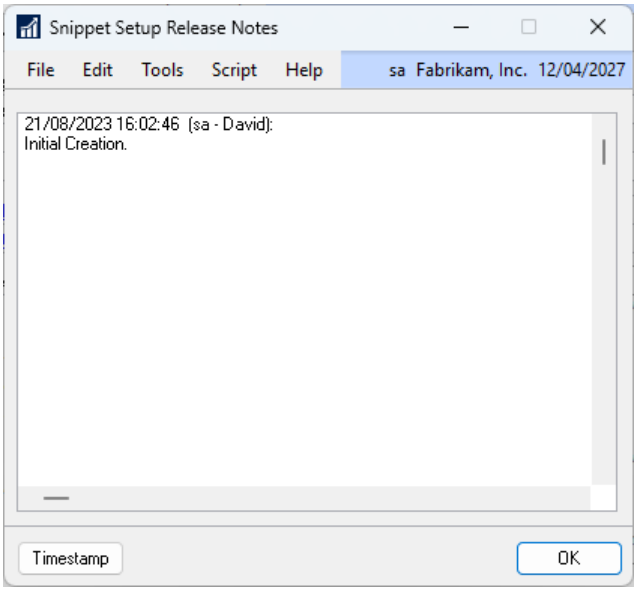
The following is a description of the individual fields on the window:

*Snippet ID*

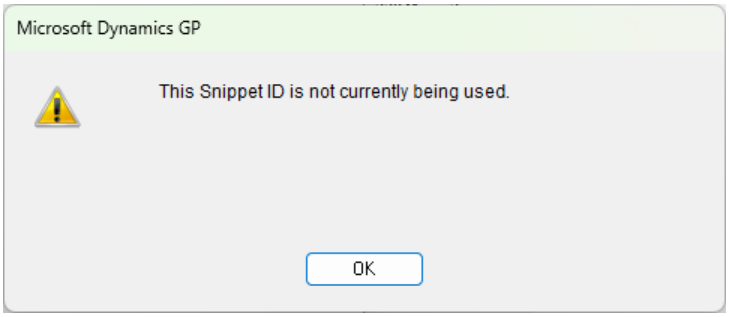
This field contains a unique identifier for each snippet in the system. The lookup button can be clicked to select from existing snippet IDs. The lookup will be filtered to the current project if the snippet belongs to a project.




The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




Click this button to display what resources are using this snippet.



*Snippet Name*

This field contains a description of the snippet.

*Snippet Mode*

This drop-down list selects the development language of the snippet. This also controls from which script editors the snippet will be available to insert.

*Snippet Menu*

This optional field allows snippets to be grouped on the Insert Button.

*Project ID*

Use this field to add the current snippet to a development project.

*Parameter ID*

Use this field to specify a Parameter List to be used with the snippet.

*Execute Dexterity SanScript code in the context of Product*

This drop-down list contains a list of products currently installed on the Microsoft Dynamics GP workstation. Only available when in Dexterity sanScript Snippet Mode.

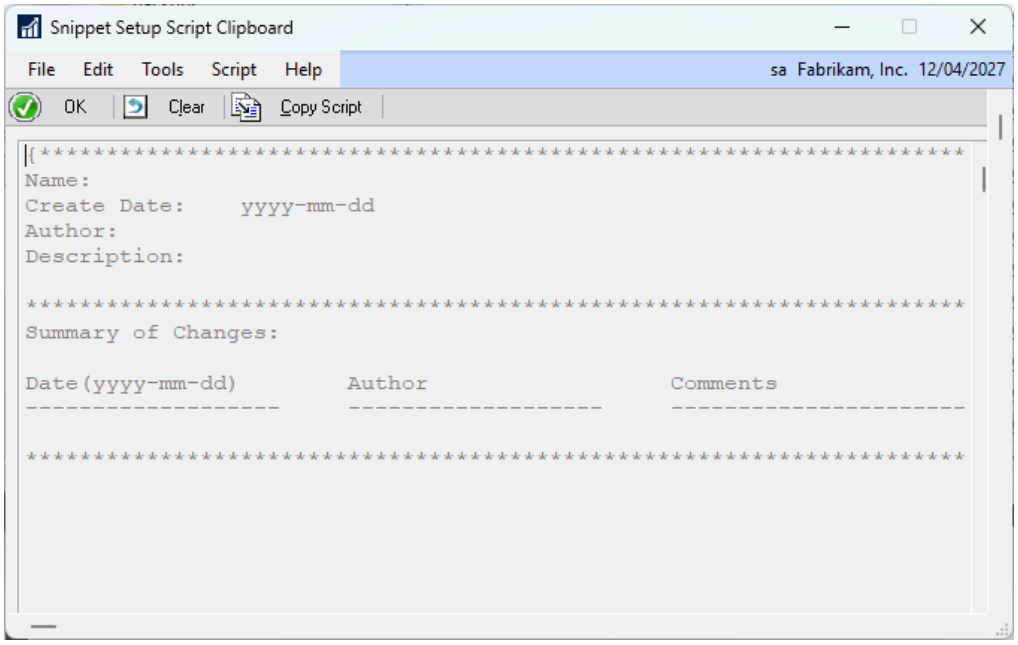
*Modified*

This checkbox can be used to identify that this snippet should be executed in the context of the modified dictionary. This allows the script to reference Modifier added local fields. Only available when in Dexterity sanScript Snippet Mode.

*To be able to execute scripts against modified dictionaries, the WinthropDC.GpPowerToolsVC.dll Addins must be installed.*

*Clipboard Button*

This expansion button opens the Snippet Script Clipboard window. This window can be used as temporary storage when editing snippets to allow for cutting and pasting between snippets. Use the Copy Script Button if you wish to copy the current snippet to the clipboard window and the Clear Script Button to clear the clipboard window contents.




*Script*
This text field contains the snippet code.

The following is a description of the additional buttons on the window:

*Help Button*
Use this button (highlighted on screenshot) to open the full Dexterity Help file. Only available when in Dexterity sanScript Snippet Mode.

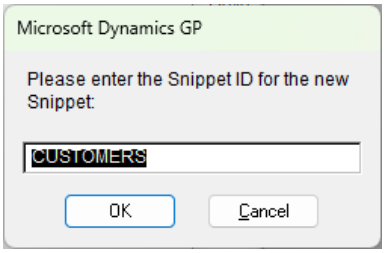
*Parameters Button*
Use this button to insert a Parameter Placeholder into the script for the Parameter List selected with the Parameter ID. See the section under Trigger Setup for more information.

*Insert Button*
Use this button to insert a code construct (dependent on development language selected) or Parameter Placeholders. See the section under Trigger Setup for more information.

*Helper Button*
Use this button to open the Insert Helper Function window and insert a helper function into the script. See the section under Trigger Setup for more information.

*Names Button*
Use this button to insert a dictionary resource into the script. See the section under Trigger Setup for more information.

*Duplicate Button*
Use this button to duplicate or rename the current snippet ID to a new script ID. This is useful when an existing script ID is very similar to the new one you want to create.

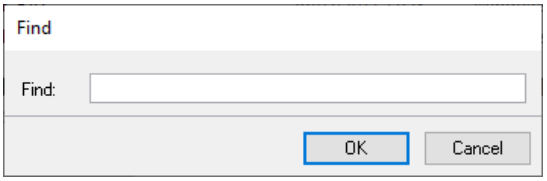



A new snippet ID must be specified in the dialog which opens.

The following is a description of the Script menu available for the window:

*Find …*

Use this menu option to open the script editor Find window to search for text. Control-F can be used as a shortcut.

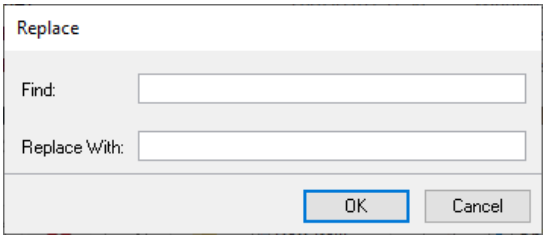



*Find Next*

Use this menu option to find the next occurrence. Control-G can be used as a shortcut.

*Replace …*

Use this menu option to open the script editor Replace window to search and replace text. Control-R can be used as a shortcut.

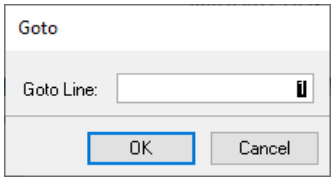



*Replace and Find Next*

Use this menu option to replace and find the next occurrence. Control-B can be used as a shortcut.

### *Goto Line …*

Use this menu option to open the script editor Goto Line window to jump to a specified line. Control-N can be used as a shortcut.




### *Save and Continue*

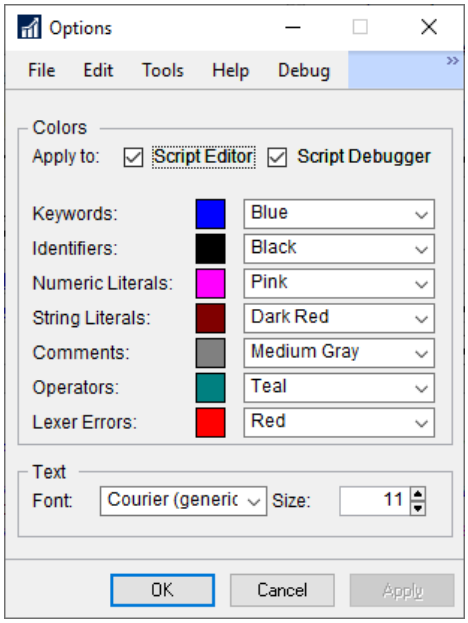
Use this menu option to save the current snippet without clearing the window. Control-S can be used as a shortcut.

### *Convert References*

Use this menu option to convert the Dexterity resource names contained in curly braces in the current snippet to their SQL equivalents. Control-0 can be used as a shortcut.

### *Options*

Use this menu option to open the Options window to allow the syntax highlighting colors, font style, and size to be changed. Control-O can be used as a shortcut.




### *Names Button Uses Clipboard*

Use this menu option to control whether the Names Button returns directly to the script (default) or to the clipboard.

*To be able to use the clipboard, the WinthropDC.GpPowerToolsVB.dll Addins must be installed.*

*Names Button Uses Fully Qualified Names*

Use this menu option to control whether the Names Button returns SQL table and columns as fully qualified, or just as table and column names.

*Names Button Adds Keyword 'Show'*

Use this menu option to control whether the Names Button automatically adds the keyword "show" to returned SQL column names so that they will be displayed as Technical Names rather than Physical Names in the SQL Query. It adds the "as" clause to column names.

## Parameter Lists

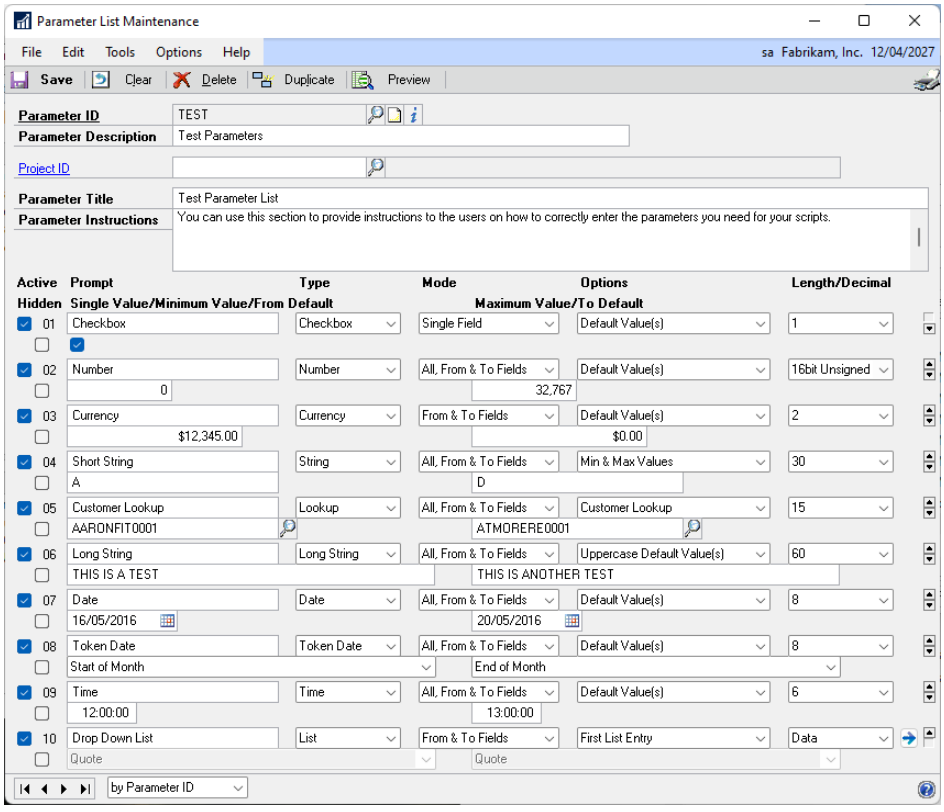
You can open the Parameter List Maintenance window by selecting Parameter Lists from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Parameter Lists from the Options button drop list on the main window. This is an Advanced Mode feature.

Parameter Lists provide a method to create a custom user interface to request information from the user prior to a script being executed. The selections made by the user can then be used in the scripts to change the behavior of the script or select the data range the script runs against.

Parameter Lists can be used with Automatic Trigger Mode Non-Logging Trigger scripts (for Focus Events, Form Menu and Field Context Menu Types), Runtime Execute Setup scripts, SQL Execute Setup scripts and .Net Execute Setup scripts.

Once a Parameter List dialog has been used, the parameters are available in the script which opened the dialog and any script called by that script as long as the called scripts are linked to the same Parameter ID or have a blank Parameter ID.

To use the data returned from the Parameter List Dialog, just insert a Parameter Placeholder into the script. This will be replaced with the data from the dialog prior to the script being executed. Parameter Placeholders are special language dependent character combinations which return the correct data type so that the script will compile and will also be recognized by the script pre-processor so they can be substituted.

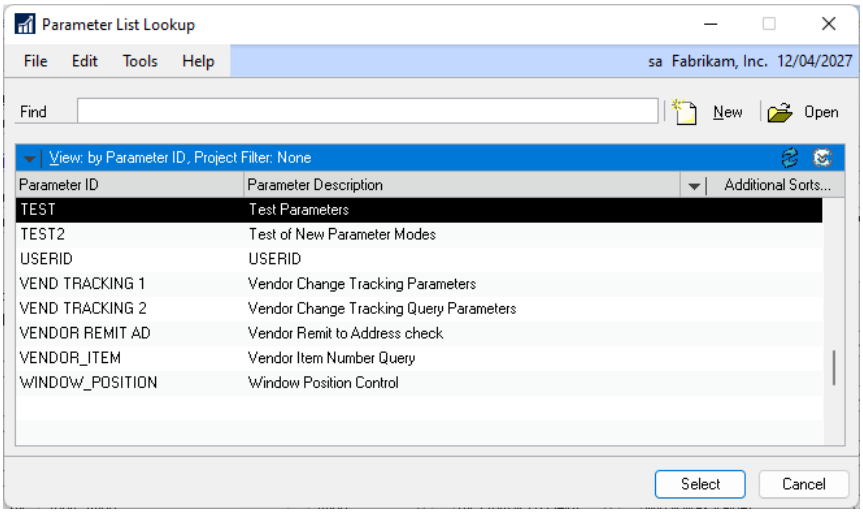


When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

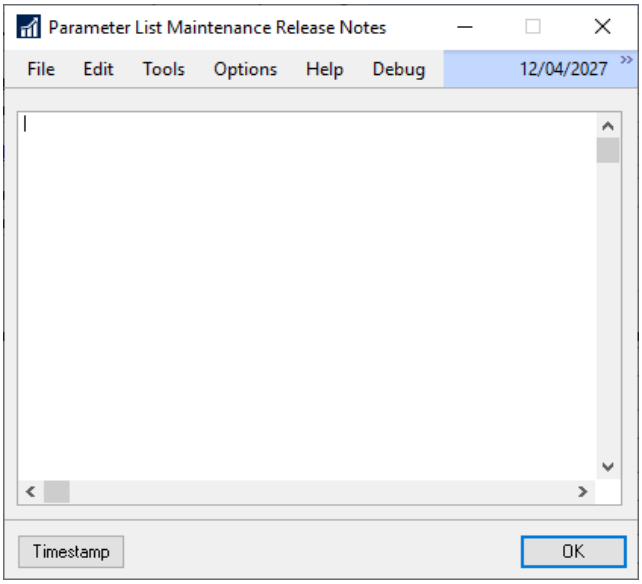
The following is a description of the individual fields on the window:

*Parameter ID*

This field contains a unique identifier for each Parameter List in the system. The lookup button can be clicked to select from existing parameter IDs. The lookup will be filtered to the current project if the parameter list belongs to a project.

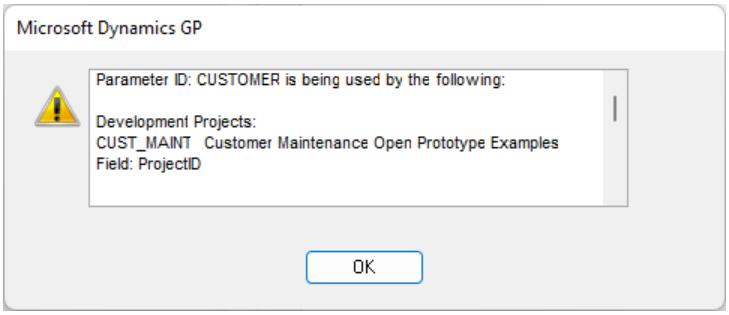



The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.



*Parameter List Information*

Click this button to display what resources are using this parameter list.




*Parameter Description*

This field contains a description of the parameter list.

*Project ID*

Use this field to add the current parameter list to a development project.

*Parameter Title*

This field contains the title for the parameter list. This value will be used at the top of the Parameter List dialog as well as for the title of the dialog's window.

*Parameter Instructions*

This field contains the instructions for the user on how they should use the parameter list. This field will be displayed at the top of the Parameter List dialog.

The following is a description of the parameter definition fields for the ten parameters on the window:

*Parameter Active*

This checkbox specifies whether the current parameter is enabled. This allows a parameter to be temporarily disabled without having to remove the rest of the settings for the parameter.

*Parameter Hidden*

This checkbox specifies whether the current parameter is hidden. This allows a parameter to be available for use with coding without being shown to the end user.

*Parameter Prompt*

This field contains a name of the parameter and will be used for the prompt of the parameter.

*Parameter Type*

Use this drop-down list to select the data type for the parameter. You can select from:

- Checkbox
- Number
- Currency
- Quantity
- String
- Lookup
- Long String
- Date
- Token Date
- Time
- List
- List (SQL)

*Parameter Mode*

This drop-down list sets whether the parameter is a single value or a range of values. You can select from:

- Single Field
- From & To Fields
- All, From & To Fields

*Parameter Options*

This drop-down list sets various options depending on the Parameter Type selected.

For Lookup Type, you can select from:

- Account Lookup
- Customer Lookup
- Customer Class Lookup
- Vendor Lookup
- Vendor Class Lookup
- Item Lookup
- Item Class Lookup
- Inventory Site Lookup
- User Lookup
- Company Address Lookup
- Employee Lookup
- Employee Class Lookup
- Salesperson Lookup
- Territory Lookup
- Shipping Method Lookup
- Payment Terms Lookup
- Price Level Lookup
- Country Code Lookup
- Checkbook Lookup
- Currency Lookup
- Custom Lookup (SQL)
- Custom Lookup (Form
- Custom Lookup (SQL) Uppercase
- Custom Lookup (Form) Uppercase

For List Type, you can select from:

- Cleared Value(s)
- First List Entry
- Default Value(s)

For other types, you can select from:

- Cleared Value(s)
- Default Value(s)
- Min & Max Values

For String and Long String Types, you can also select from:

- Uppercase Cleared Value(s)
- Uppercase Default Value(s)
- Uppercase Min & Max Values

*Parameter Length/Decimal*

This drop-down list controls the length of the field or the number of decimal places for the parameter fields.

For Number Type, you can select the size of the parameter 16 or 32 bit and whether the number should be signed or unsigned.

For Currency and Quantity Types, you can select the number of decimal places from 0 to 5.

For String and Long String Types, you can select the length of the string in 5 character increments.

For List Type, you can select what will be returned by the parameter, you can select from:

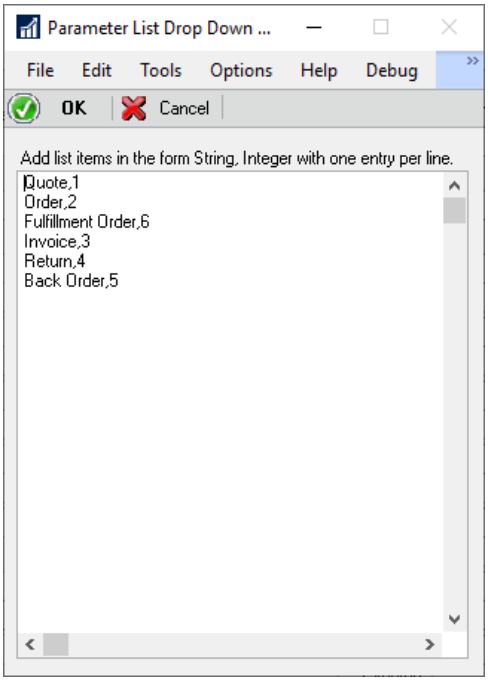
- Position
- Data
- String

For all other Types, the length value is fixed.

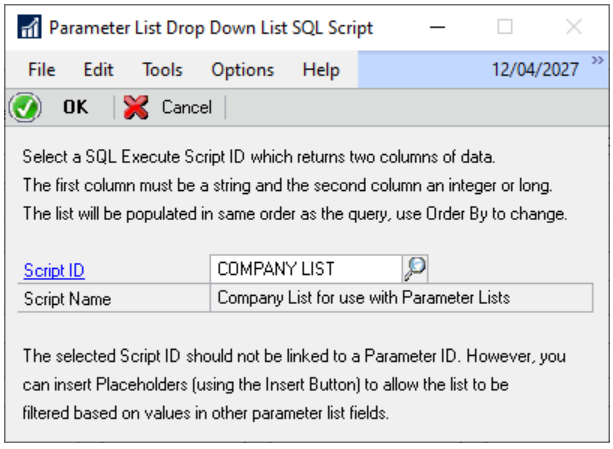
*Parameter Expansion Button*

Click this button to open the appropriate Parameter List Maintenance Additional Information window.

For Drop-down lists, it will open the Parameter List Drop-down List Maintenance window. This window is used to set up the drop-down list values for a List Type parameter. The drop-down list will be populated with the string on each line in the order it is listed in the window. You can specify an integer (32 bit) value for each entry using a comma.

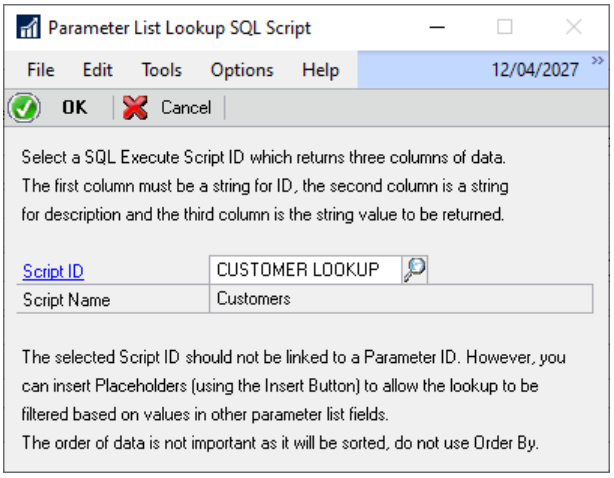



For SQL Drop-down lists, it will open the Parameter List Drop-down List SQL Script window. This window is used to select a SQL Execute Setup script which returns an ordered query with two columns; a string and an integer (32 bit) value.



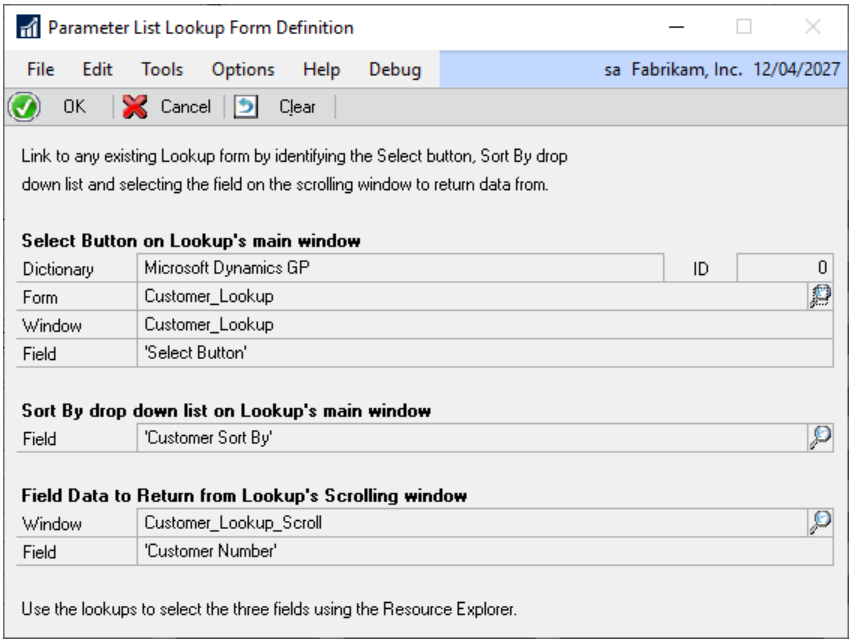

*The SQL Execute script used to populate the SQL List can include placeholders which will be substituted with values from other parameter list fields when the script is executed. This allows to contents of the list to be changed based on other parameter data already entered.*

For SQL Custom Lookups, it will open the Parameter List Lookup SQL Script window. This window is used to select a SQL Execute Setup script which returns a query with three string columns; an ID string, a Description string and a string to be returned (usually the same as the ID value).




*The SQL Execute script used to populate the SQL Lookup can include placeholders which will be substituted with values from other parameter list fields when the script is executed. This allows to contents of the lookup to be changed based on other parameter data already entered.*

For Form based Custom Lookups, it will open the Parameter List Lookup Form Definition window. This window is used to define the form, window and field information required to drive an existing lookup form in any dictionary installed in Microsoft Dynamics GP.




### *Parameter Single/Minimum/From Value*

Depending on the Parameter Option selected, this field can be used to specify a Minimum value for the parameter, or a default value for the Single or From field of the parameter.

### *Parameter Maximum/To Value*

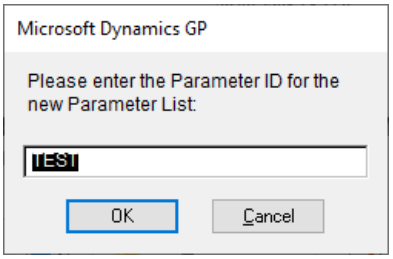
Depending on the Parameter Option selected, this field can be used to specify a Maximum value for the parameter, or a default value for the To field of the parameter.

*The order of the ten parameter in the Parameter List can be adjusted using the small up buttons and down buttons on the right-hand side of the window.*

The following is a description of the additional buttons on the window:

### *Duplicate Button*

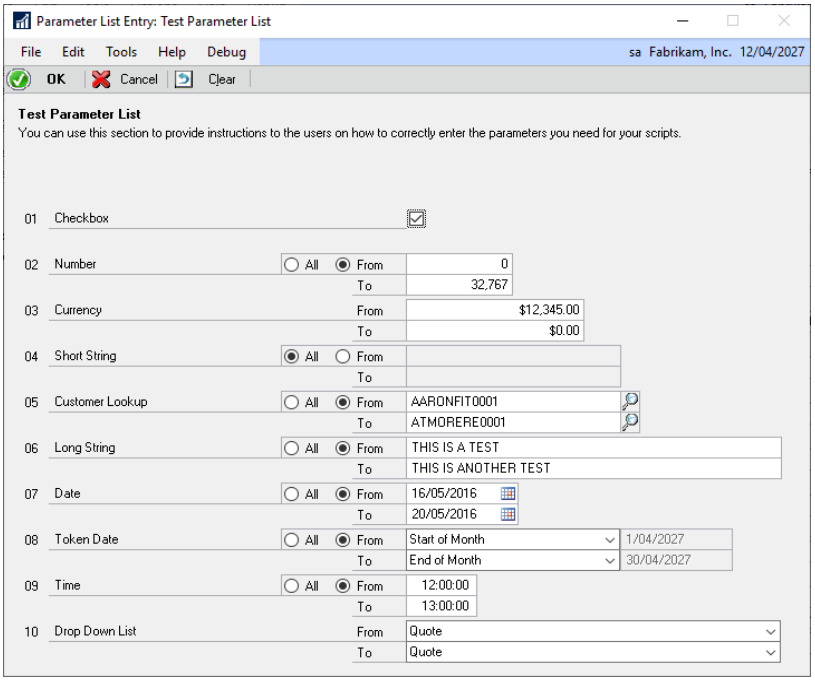
Use this button to duplicate or rename the current parameter ID to a new parameter ID. This is useful when an existing parameter list is very similar to the new one you want to create.




A new parameter ID must be specified in the dialog which opens.

### *Preview Button*

Use this button to show a preview of what the Parameter List dialog will look like when it executed. If the OK button is used to close the Parameter List dialog, an informational dialog will open showing the results returned for the various languages.



*The Parameter List dialog window will automatically resize to the size needed to display only the used and active parameters. If a parameter is unused or not active it will leave a gap in the dialog, except at the bottom where the window will be resized smaller.*

The following is a description of the Options menu available:

### *Save and Continue*

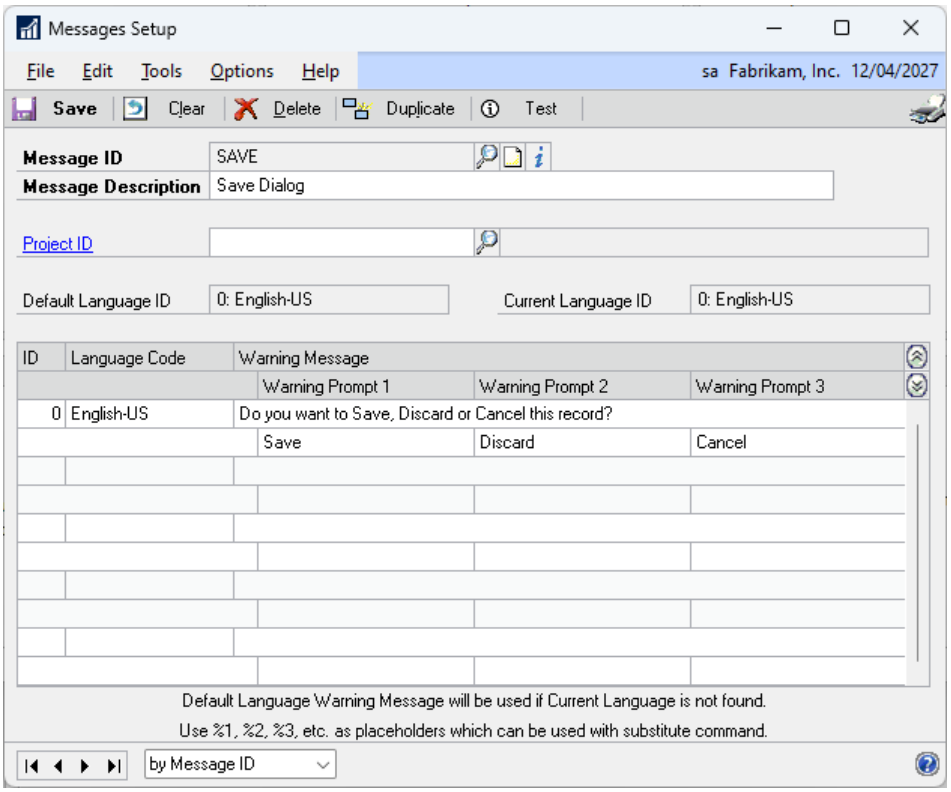
Use this menu option to save the current parameter list without clearing the window. Control-S can be used as a shortcut.

## Messages Setup

You can open the Message Setup window by selecting Messages Setup from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Messages Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

Messages Setup allows you to define reusable warning message text in multiple languages which can be used in the Trigger Setup window or with Helper Functions.

*Using Messages allows the same message to be displayed in multiple locations and for the message to be automatically translated on multi-lingual systems. If a change to the message is required, it can be updated in a single location as it is not hard coded into scripts.*

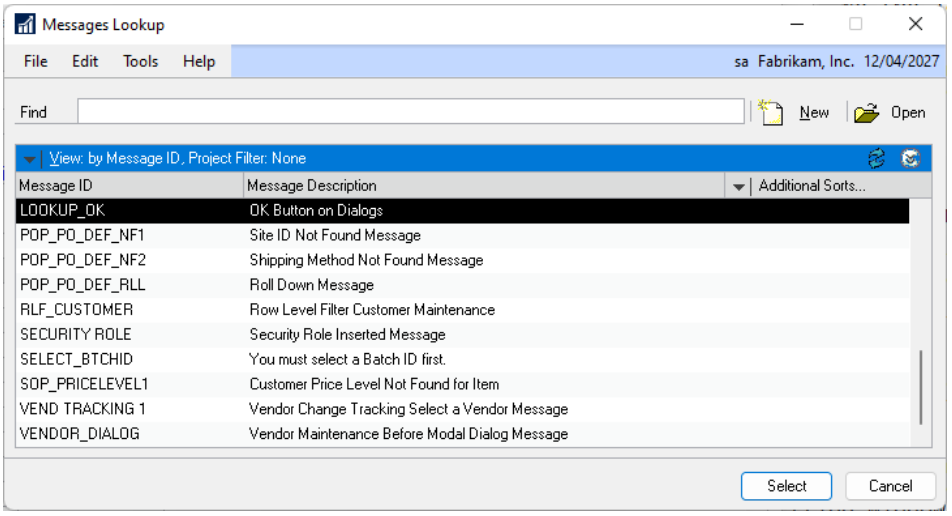



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

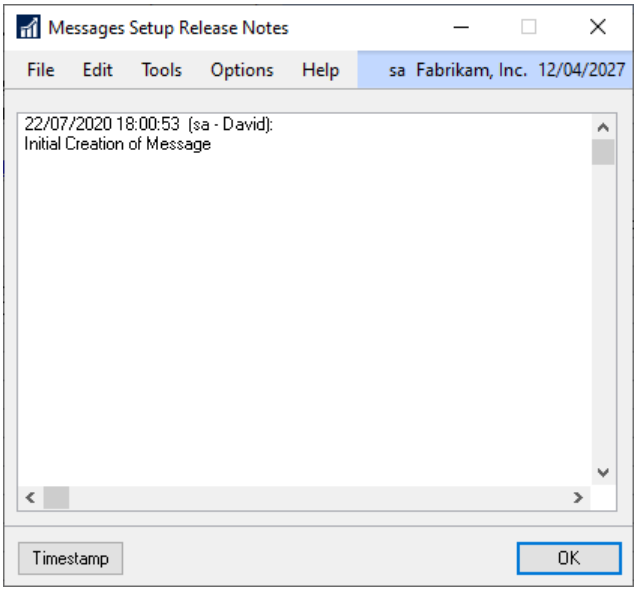
The following is a description of the individual fields on the window:

*Message ID*

This field contains a unique identifier for each Message in the system. The lookup button can be clicked to select from existing message IDs. The lookup will be filtered to the current project if the message belongs to a project.



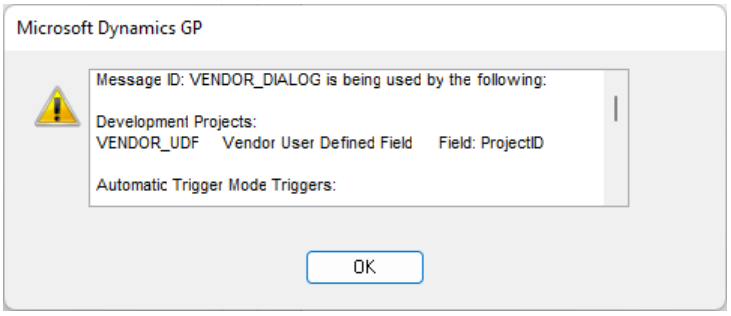

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Message IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

*Messages Information*

Click this button to display what resources are using this message.




*Description*

This field contains a description of the message.

*Project ID*

Use this field to add the current message to a development project.

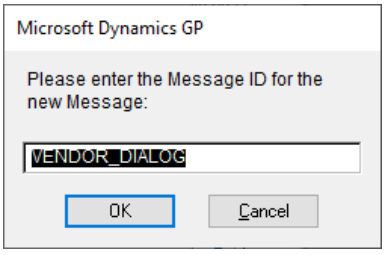
*Message List*

This scrolling window is where you can enter the message and 3 prompts for buttons for each language. If there is no message for a particular language, the message for the default language will be used instead.

The following is a description of the additional buttons on the window:

*Duplicate Button*

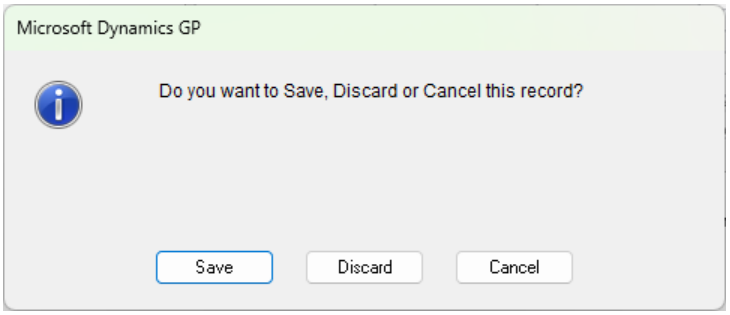
Use this button to duplicate or rename the current message ID to a new message ID. This is useful when an existing message ID is very similar to the new one you want to create.




A new message ID must be specified in the dialog which opens.

### *Test Button*

Use this button to display a system dialog based on the message ID currently displayed.



## Dynamic Trigger Logging

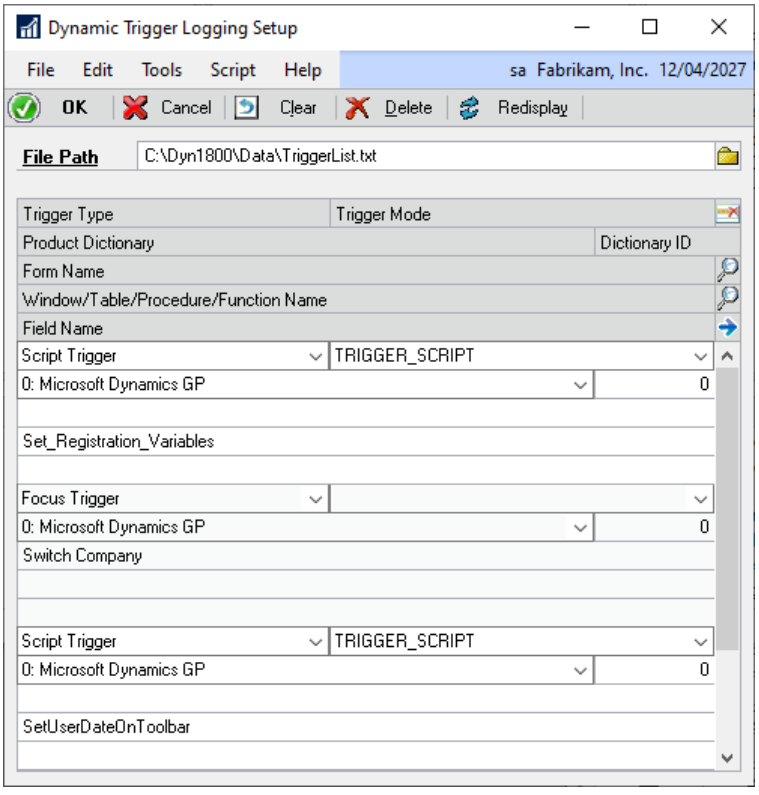
You can open the Dynamic Trigger Logging window by selecting Dynamic Trigger Logging from the Cards section of the GP Power Tools Area Page or by selecting Scripting >> Dynamic Trigger Logging from the Options button drop list on the main window. This is an Advanced Mode feature.

There are times when you are unable to use convention Dexterity Script Logging to follow the flow of scripts in Microsoft Dynamics GP. Some examples are:

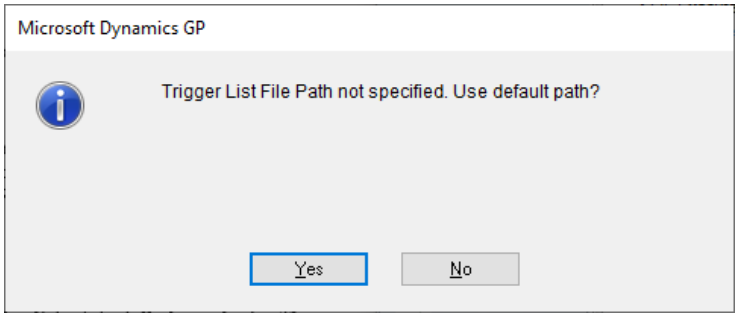
- Dexterity Script Logging is unavailable when using Service Based Architecture (SBA) and cannot be enabled.
- Dexterity Script Logging sometimes causes instability which can cause Microsoft Dynamics GP to crash.

Dynamic Trigger Logging can be used as an alternative method to track the flow of scripts. By registering triggers before and after any event (Focus, Table or Script) in the system and logging when that trigger fires, you can track when code is executed.

*You will need to know the focus events, table names and script names in advance to be able to register dynamic triggers against them. When working with a Service Procedure for Service Based Architecture, you could capture logs of the code running in the desktop client to get all the procedure and function names. Once you have the names, they can be used to set up the dynamic triggers.*



*When you first open the Dynamic Trigger Logging window, if the file path has not been written into the Dex.ini file, you will be asked if you want to use the default setup file name.*




The following is a description of the individual fields on the window:

*File Path*
This field contains the path to the Dynamic Trigger Logging setup file.

*Trigger Type*
Select the type of trigger from Focus Trigger, Table Trigger or Script Trigger.

*Trigger Mode*
Select the mode of the trigger depending on the type. Focus Triggers can use Pre, Change, Post, Print, Activate, Fill, Insert and Delete. Table Triggers can use Read, Read Lock, Read Both, Add, Update, Save and Delete. Script Triggers use a single Script mode for both Procedures and Functions.

*Product Dictionary*
Select a product dictionary from the list of installed products. The Dictionary ID field will be updated automatically.

*Dictionary ID*
Select a product using its dictionary ID. The Product Dictionary field will be updated automatically.

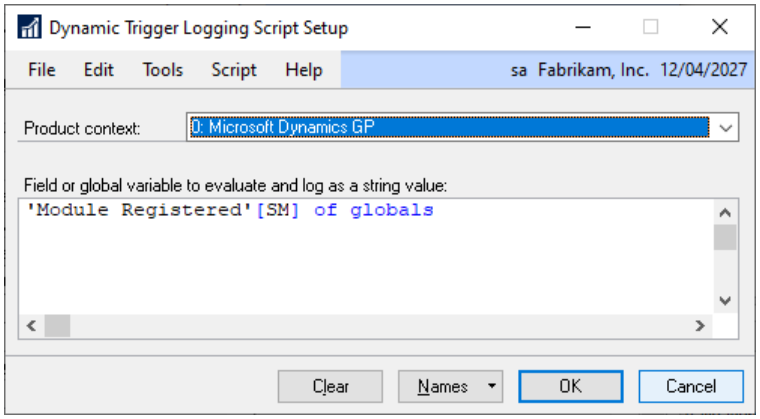
*Form Name*
Enter the form name or use the lookup to select. This field is required for Focus Triggers and is optional for Table Triggers and Script Triggers.

*Window/Table/Procedure/Function Name*
Enter the window, table, procedure or function name as appropriate or use the lookup to select. Functions are denoted by ending with "()".

*Field Name*
Enter the field name or use the window lookup to select. Adding a field name to a Focus Trigger or Table Trigger will get the value of the field displayed in the log entries created.

### *Script Expansion Button*

For each trigger created, clocking the expansion button will allow an additional expression to be entered as a script. This expression will be evaluated, and the resulting data will be logged with the trigger.




The expression script must not contain a semicolon. It can be used to display a field value off a window or a global variable value. Use the Names button to search for the desired field or global variable.

The following is a description of the additional buttons on the window:

### *OK Button*

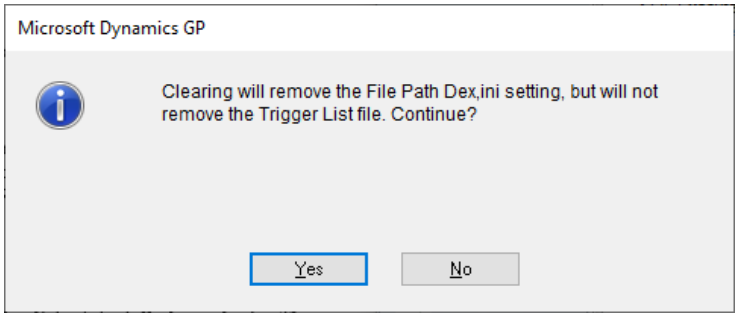
This button will save the triggers to the Dynamic Trigger Logging setup file as defined by the File Path field and writes the File Path into the MBS_Debug_LogListPath Dex.ini Setting.

### *Cancel Button*

This button will close the window without making any changes.

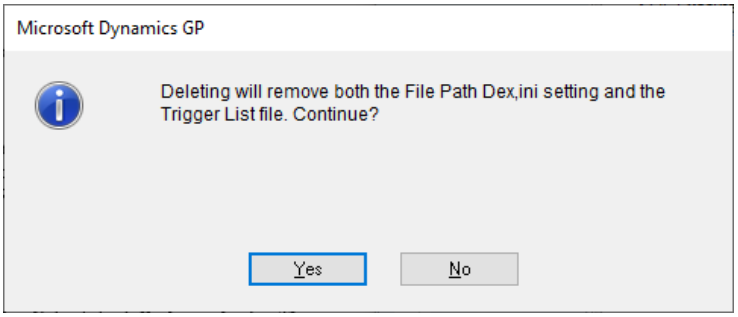
### *Clear Button*

Use this to clear the window. Note it will only clear the File Path field and remove the MBS_Debug_LogListPath Dex.ini Setting.



### *Delete Button*

Use this clear the window. Note it will clear the File Path field and remove the MBS_Debug_LogListPath Dex.ini Setting and delete the Dynamic Trigger Logging setup file (if it exists).




### *Redisplay Button*

Use this button to redisplay the list of triggers and scroll to the bottom of the list ready to add a new Trigger.

When you launch Microsoft Dynamics GP, the MBS_Debug_LogListPath Dex.ini Setting is checked and if it contains a path valid setup file, Dynamic Trigger Logging will parse the setup file and register the triggers specified. When triggers fire, they will write a record in the GP Power Tools log files.

*The Dynamic Trigger Logging setup file is a text file and can be edited manually outside of Microsoft Dynamics GP using your favorite text editor or Notepad.exe. The format is explained in the section at the end of the GPPTools.txt file installed with GP Power Tools.*

*While the triggers registered by Dynamics Trigger Logging do not perform any function other than writing a log entry, it is recommended that this feature is only used by Dexterity developers.*

# Virtual Fields

GP Power Tools Build 31 or later adds Virtual Field functionality which can be used to programmatically add fields to windows without needing Modified or Alternate windows.

## Introduction

Virtual Field functionality has existed in Dexterity since version 8.0, but it was never documented and released publicly. The core Virtual Field functionality was very limited as it was originally designed to only work with read only fields in a scrolling window for the original Dexterity based Navigation Lists. More recently Virtual Fields were used with the Dexterity based Workflow implementation.

Using the recently added improvements to Virtual Fields, combined with custom code, GP Power Tools now offers Virtual Field functionality by a series of Helper Functions and a Runtime Execute Custom Script Purpose.

## Adding Virtual Fields

Virtual Fields can be added to any window in any dictionary in Microsoft Dynamics GP and can be based on any existing field defined in any dictionary, assuming that field is not already on the window.

The Virtual Fields must be added using a Trigger Setup Form Pre trigger running before original script, or Form Control Reject Form Pre Script, or from a Runtime Execute script called from one of those events. Calling the Virtual Field Helper Functions from any other location will generate an error message in the GP Power Tools log and abort.

*While you can use a dictionary field from any dictionary, GP Power Tools has a number of fields added to its dictionary for use with the Virtual Field functionality. These fields are named with VF_Field_<Type>_XX convention in the dictionary and support about 35 datatypes/keyable length/format combinations. There are 10 of each of these fields available numbered from 01 to 10. It is recommend to use these fields to ensure that they are fields that will never exist on any window.*

When adding Virtual Fields, you have the options to add a Prompt, a Lookup and format field to the Virtual Field. Choose the MBS_Add_Virtual_Field Helper Function that has the features you need.

## Addressing Virtual Fields

As Virtual Fields do not actually exist on a form definition in a dictionary you cannot address them directly from Dexterity. You will need to use a Helper Function to get a reference (pointer) to the field and then use Helper Functions to work with the field or use the *field()* function to re-instantiate the reference.

*Using the field() function to re-instantiate the reference will only work if the script is executed in the same dictionary context as the form and window that the field exists on.*

## Triggering On Virtual Fields

When using the Helper Functions to add Virtual Fields to windows in the core Dynamics.dic dictionary, you have the option to register Field Pre, Field Change, and Field Post triggers against the Virtual Field and its optional Lookup.

Once the triggers have been registered, you can use a Runtime Execute script with the 5261: VirtualFieldHandler Script Purpose to take any desired actions when the triggers fire.

*Virtual Field triggers do not work with third party dictionaries. It is recommended to either only use read only fields or to perform any validation needed for editable fields when saving the record. Using Lookups or button fields will not work as it is not possible to trigger a script.*

## Making Space for Virtual Fields

Virtual fields can be added to any location on a window. They can be added to a blank space if one is available or over the top of an existing field, if desired. If you wish to expand a window to make space for additional Virtual Fields, there is an MBS_Expand_Virtual_Field_Window Helper Function.

Horizontal lines can also be added using the MBS_Add_Virtual_FieldLine Helper Function.

## Virtual Field Limitations

Due to undocumented nature of Virtual Fields, there are some limitations caused by the underlying Dexterity language:

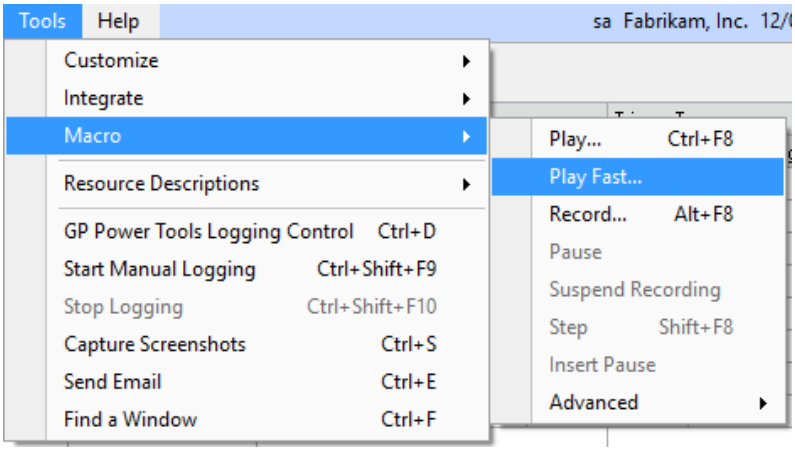
- Virtual Fields do not work with the Web Client.
- Adding Triggers to Virtual Fields only works for windows contained in the core Dynamics.dic dictionary.
- Virtual Fields have black borders. This is because Virtual Fields were originally created to work in scrolling windows where the border should not be shown. Using a black border was the only way to show a border at all.

## Additional Developer Features

GP Power Tools adds some extra features to help developers. Below is a summary of the features:

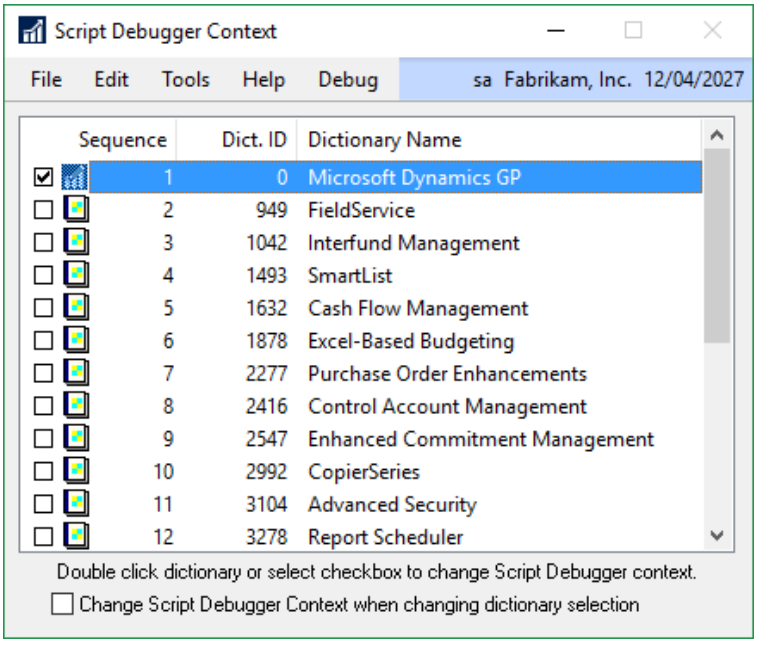
### Macro Play Fast

Added to the Macro menu is the option to Play Fast. This option is the same as the normal play macro option but runs about three times faster.



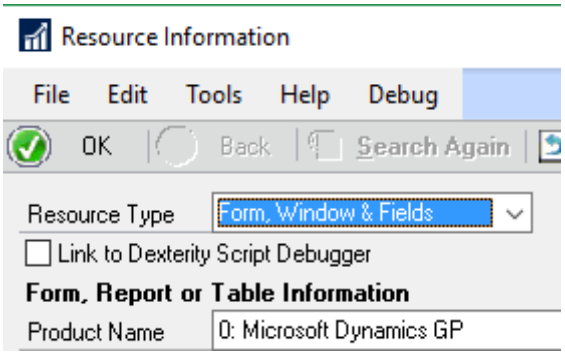

### Script Debugger Context

When the Dexterity Script Debugger is opened, the Script Debugger Context window is opened automatically. This window can be used to change the Script Debugger Dictionary Context easily without needing change Dex.ini settings or restart the application. Use the checkbox at the bottom of the window to control if single or double click is need to change context.



## Resource Information Context

When the Dexterity Script Debugger is enabled, the Resource Information window (when in Form, Window & Fields mode) has a Link to Dexterity Script Debugger option which will link the Dictionary drop down list on the Resource Information window to Script Debugger Context.




## Runtime Execute Context

When the Dexterity Script Debugger is enabled, the Runtime Execute Setup window will default the Dictionary Context to match the Script Debugger Context.

## Open Script Debugger on Startup

When the Dexterity Script Debugger is enabled, the MBS_Debug_Break Dex.ini Setting can be used to force the Script Debugger to open upon starting Microsoft Dynamics GP. You can then use the Script Debugger Context window to change Dictionary context and the Script Debugger window to set breakpoints.

# Chapter 6: Form Control Tools Features

This chapter includes the following sections:

- *Form Control*
- *Form Control Setup**
- *Password Setup**
- *Form Control Status**
- *Form Control Resources**

** Advanced Mode Feature*

# Form Control

GP Power Tools Form Control is a "No Code / Low Code" customization solution for Microsoft Dynamics GP. It uses resource filters and rule based settings to perform many different functions on any form, in any Dexterity product dictionary, within Microsoft Dynamics GP.

It has over 50 rule types which can be applied at the Form Level, Window Level, Scrolling Window Level and Field Level.

Form Control can be used to completely replace and improve upon Microsoft Dynamics GP Field Level Security (which was originally created by Winthrop before being acquired by Microsoft).

It also has rules specifically designed to be used with the Developer Tools module, to allow single scripts to be triggered across multiple forms, instead of needing to create separate triggers for every form.

## Introduction

Form Control works when a form is opened by checking if any Form Control IDs have a base settings resource filter that includes the current form. If there is at least one Form Control ID that needs to be applied, Form Control will read the dictionary and cache the Form, Window/Scrolling Window and Field resources so it has a full list of the resources on the form.

Form Control will then check the resource filters for each rule and if met, it will apply the rule to the resources.

Every rule can have a Password associated with it, which can control whether the rule should be applied or not. Passwords are defined using the Password Setup window.

If Developer Tools is registered, every rule can also have a Form Control Condition Script associated, which can control whether the rule should be applied, reversed or not applied. Form Control Conditional Scripts are created using the Runtime Execute Setup window using the 5261: FormControlConditional Script Purpose.

Once Form Control decides a rule should be applied, it will register triggers on the appropriate events to execute the rule. The list of triggers registered can be reviewed from the Form Control Status window and the list of resources cached for forms can be reviewed from the Form Control Resources window.

*Form Control Settings are applied automatically across the system as windows are opened. If settings have been changed which affect a window while that window is already open, the Form Control settings will be re-applied when the window is next opened.*

Form Control can also be used with Virtual Fields to add additional fields to multiple windows with a single rule.

## Form Control Rule Types

There are over 60 Rule types for Form Control. A summary of each rule is provided below. The rules are classified in three types:

- **Field Level Security rules** for replacing Field Level Security. Form Control is more powerful and works in situations where Field Level Security fails. It is easier to maintain, can work across multiple forms, windows and fields, and is compatible with the Task and Role Security model.
- **Form Control rules** for adding "No Code / Low Code" customizations to forms, windows and fields in Microsoft Dynamics GP.
- **Developer Tools rules** for replacing multiple Developer Tools Triggers created using the Trigger Setup window with Form Control triggers that can work across multiple forms, window and fields. These rules are designed to be used with a Form Control Conditional Script.

*Note that all rule types can be controlled by adding a Password ID or Script ID. When using a script, it is also possible to programmatically change the Warning Message and Expression fields.*

### Form Rules

The section below covers form level rules:

*Password Form*

This Field Level Security rule allows a password to be requested before access to a form is granted.

*Disable Form*

This Field Level Security rule allows access to a form to be denied. It is similar to denying access with security (or Deny based Security) but can be conditional based on a script.

*Form Menu Shortcut*

This Form Control rule can be used to add a menu entry with an optional keyboard shortcut to any form. The rule can run the change script for any field on the window (usually a button). Fields added to the Key Field List will have their values stored before running the change script and restored afterwards. This rule can be used to add Save and Continue or Print and Continue functionality to windows.

*Reject Form Pre Script*

This Developer Tools rule allows a Form Pre Before Original trigger to be registered across multiple forms, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Form Pre Script*

This Developer Tools rule allows a Form Pre After Original trigger to be registered across multiple forms, and call a single Form Control Conditional Script.

*Reject Form Post Script*

This Developer Tools rule allows a Form Post Before Original trigger to be registered across multiple forms, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Form Post Script*

This Developer Tools rule allows a Form Post After Original trigger to be registered across multiple forms, and call a single Form Control Conditional Script.

### Window Rules

The section below covers window level rules:

*Password Window*

This Field Level Security rule allows a password to be requested before access to a window is granted.

*Disable Window*

This Field Level Security rule allows access to a window to be denied. This provides more granular security than standard security.

*Change Window Title*

This Form Control rule can be used to replace the title of a window, or perform a find and replace on the window title, using the specified semicolon (;) separated terms in the Expression field.

*Clear Changes Before Window Close*

This Form Control rule will clear the "changed" flag on a form before the window closes, thus preventing the "Do you want to Save?" dialog from appearing. It is used to block saving on a window. The Key Field List can specify any status fields that also need to be cleared.

*Focus First Window Field*

This Form Control rule can be used to set the focus to the first editable field in the last window specified in the resource filter. It is recommended to only have a single window in the resource filter.

*Reject Window Pre Script*

This Developer Tools rule allows a Window Pre Before Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Window Pre Script*

This Developer Tools rule allows a Window Pre After Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script.

*Reject Window Post Script*

This Developer Tools rule allows a Window Post Before Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Window Post Script*

This Developer Tools rule allows a Window Post After Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script.

*Reject Window Activate Script*

This Developer Tools rule allows a Window Activate Before Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Window Activate Script*

This Developer Tools rule allows a Window Activate After Original trigger to be registered across multiple forms and windows, and call a single Form Control Conditional Script.

### Scrolling Window Rules

The section below covers scrolling window level rules:

*Lock Scrolling Window*

This Form Control rule will change the state of a Scrolling window to read only. Thus, preventing any changes to the window or adding new records. Note that Insert and Delete functionality must be blocked separately using the Reject Scrolling Window Insert and Reject Scrolling Window Delete rule types.

*Reject Scrolling Window Save*

This Developer Tools rule allows a Scrolling Window Change Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing, thus preventing a table row being saved.

*After Scrolling Window Save*

This Developer Tools rule allows a Scrolling Window Change After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

*Reject Scrolling Window Delete*

This Developer Tools rule allows a Scrolling Window Delete Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing, thus preventing a table row being deleted.

*After Scrolling Window Delete*

This Developer Tools rule allows a Scrolling Window Delete After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

*Reject Scrolling Window Insert*

This Developer Tools rule allows a Scrolling Window Insert Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing, thus preventing a table row being inserted.

*After Scrolling Window Insert*

This Developer Tools rule allows a Scrolling Window Insert After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

*Reject Scrolling Window Fill*

This Developer Tools rule allows a Scrolling Window Fill Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing, thus preventing a table row being displayed.

*After Scrolling Window Fill*

This Developer Tools rule allows a Scrolling Window Fill After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

*Reject Scrolling Window Pre*

This Developer Tools rule allows a Scrolling Window Pre Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Scrolling Window Pre*

This Developer Tools rule allows a Scrolling Window Pre After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

*Reject Scrolling Window Post*

This Developer Tools rule allows a Scrolling Window Post Before Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Scrolling Window Post*

This Developer Tools rule allows a Scrolling Window Post After Original trigger to be registered across multiple forms and scrolling windows, and call a single Form Control Conditional Script.

### Field Rules

The section below covers field level rules:

*Password Field Before*

This Field Level Security rule allows a password to be requested before allowing a field to gain focus. For buttons use the Password Field After rule.

*Password Field After*

This Field Level Security rule allows a password to be requested after a change is made (or button clicked). If the password is not answered correctly the change will be reversed and the field (or button) change script aborted.

*Warning Field Before*

This Field Level Security rule displays an optional warning and prevents a field gaining focus. It does not work with buttons.

*Lock Field*

This Field Level Security rule will lock a field which will prevent editing but does not grey out the field. It is applied using the Window Pre and Window Activate events and Field Value Changed event for any fields in the Key Field List. It can also be added directly to the Field Pre event for each field. Using the reverse option with a Form Control Conditional Script you can lock and unlock fields based on code.

*Disable Field*

This Field Level Security rule will disable a field which will prevent editing and greys out the field. It is applied using the Window Pre and Window Activate events and Field Value Changed event for any fields in the Key Field List. It can also be added directly to the Field Pre event for each field. Using the reverse option with a Form Control Conditional Script you can disable and enable fields based on code.

*Hide Field*

This Field Level Security rule will hide a field from a window. It is applied using the Window Pre and Window Activate events and Field Value Changed event for any fields in the Key Field List. It can also be added directly to the Field Pre event for each field. Using the reverse option with a Form Control Conditional Script you can hide and show fields based on code.

*Default Field Value*

This Form Control rule can set the default value for a field based on the Expression field. It will override any value when the form is restarted but will only insert a value when the field gains focus if it is empty.

*Clear Field Value*

This Form Control rule will clear a field value.

*Set Field Value*

This Form Control rule will set a field value based on the Expression Field.

*Strip Invalid Field Characters*

This Form Control rule will strip invalid characters from a field. Valid characters are provided using the Expression Field. Only valid characters will remain in the field. Leading and trailing spaces are also removed. This rule is designed to avoid entry of special characters into key fields, such as Customer ID, Vendor ID and Item Number.

*Validate Field Value*

This Form Control rule will validate the data in a field and display a warning and restore the previous data if the validation fails. The semicolon (;) separated list of validation rules is provided in the Expression Field. It can even be provided as a Regular Expressions (RegEx).

### *Format String Field Value*

This Form Control rule will apply a format to field and display a warning if the format cannot be matched. The format is provided by the Expression field and the warning by the Warning Message field. Using a Form Control Conditional Script allows both the Expression and Warning Message to be programmatically changed to provide conditional formatting. This can be used to match different phone number or tax ID number formats based on what was entered.

### *Mask Field Value*

This Form Control rule will change the background and font color of a field to mask the data contained in the field. The masking is removed, and the data becomes visible when the field gains focus. Using the reverse option with a Form Control Conditional Script you can mask and unmask fields based on code. Using a Password Field Before rule as well can prevent unmasking a field unless the appropriate password is entered.

### *Uppercase Field Value*

This Form Control rule will force a string field into uppercase.

### *Set Field Background Color*

This Form Control rule will change the background color of a field as a visual cue to highlight the field. Using the reverse option with a Form Control Conditional Script you can change and reset the color based on code.

### *Set Field Font Color*

This Form Control rule will change the font color of a field as a visual cue to highlight the field. Using the reverse option with a Form Control Conditional Script you can change and reset the color based on code.

### *Add Required Field*

This Form Control rule will mark a field as required. It will not work if the underlying code in the window does not check the Dexterity *required()* function. It also will not work if applied to fields on windows that are not opened.

### *Enable Autocomplete Field*

This Form Control rule can be used to enable AutoComplete for a field. If desired, you can preload values specified as a semicolon (;) separated list in the Expression Field. You can also use the Validate Field Value if you want to limit the values entered using the list keyword.

### *Round Decimals Field Value*

This Form Control rule will round a numeric field to the number of decimal places specified in the Expressions field. The number of decimal places can either be directly specified or relative to Currency or Quantity Decimal Places from an Item in the Item Master table with the Item Number field specified in the Key Field List.

It will not change the visible number decimal places as that will cause rounding issues elsewhere in the application.

*Change Field Caption*

This Form Control rule can be used to replace the caption of a field, or perform a find and replace on the field caption, using the specified semicolon (;) separated terms in the Expression field.

*Set Focus to Field*

This Form Control rule can be used to set the focus to the last field specified in the resource filter. It is recommended to only have a single field in the resource filter.

*Set Focus to Next Field*

This Form Control rule can be used to set the focus to the next editable field after the last field specified in the resource filter. It is recommended to only have a single field in the resource filter.

*Clear Changes Before Field*

This Form Control rule will clear the "changed" flag on a form before the field script executes, thus preventing the "Do you want to Save?" dialog from appearing. It is used to block saving on a window. This should be applied to fields that can change the record displayed such as the primary key lookup button or the browse buttons.

*Reject Field Pre Script*

This Developer Tools rule allows a Field Pre Before Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Field Pre Script*

This Developer Tools rule allows a Field Pre After Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script.

*Reject Field Change Script*

This Developer Tools rule allows a Field Change Before Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Field Change Script*

This Developer Tools rule allows a Field Change After Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script.

*Reject Field Post Script*

This Developer Tools rule allows a Field Post Before Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script. The returned value from the script can stop the original code from executing.

*After Field Post Script*

This Developer Tools rule allows a Field Post After Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script.

*After Field Value Changed Script*

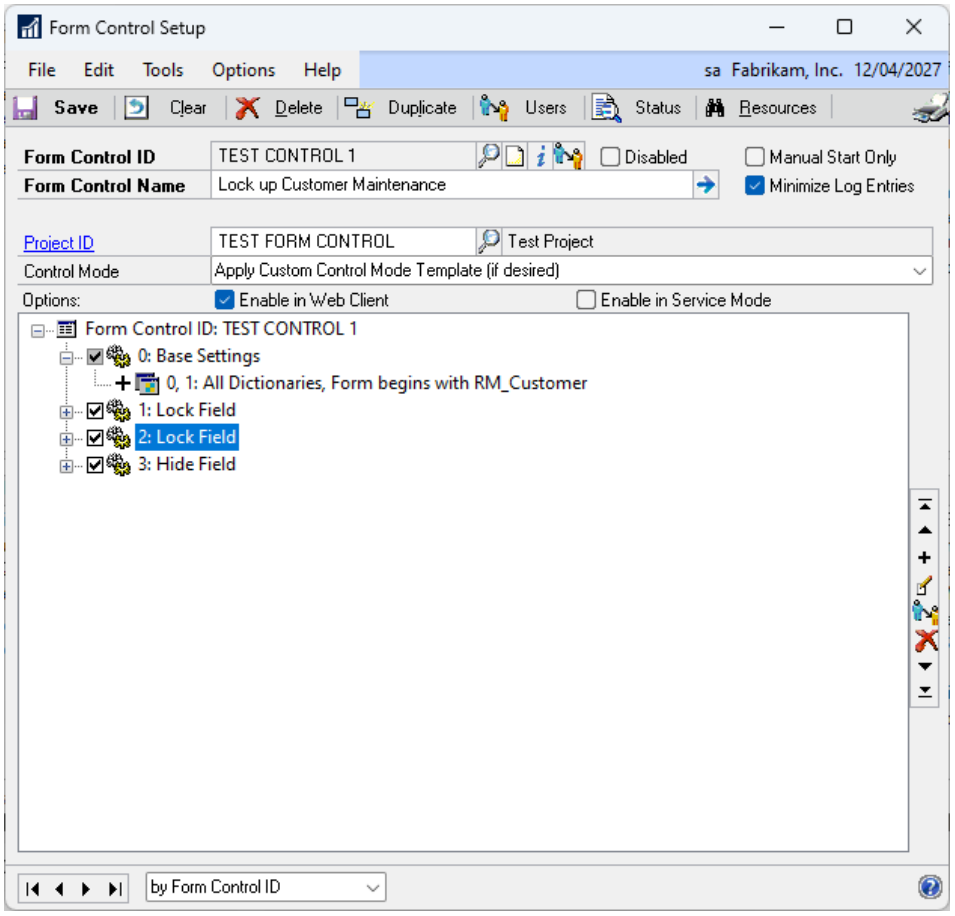
This Developer Tools rule allows a Field Value Changed After Original trigger to be registered across multiple forms, windows and fields, and call a single Form Control Conditional Script.

## Labels

You can add label rules to a Form Control ID. These labels have no functionality but can be used to break rules into groups by adding blank labels and to label those groups, if desired.

## Form Control Setup

You can open the Form Control Setup window by selecting Form Control Setup from the Cards section of the GP Power Tools Area Page or by selecting Form Control Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

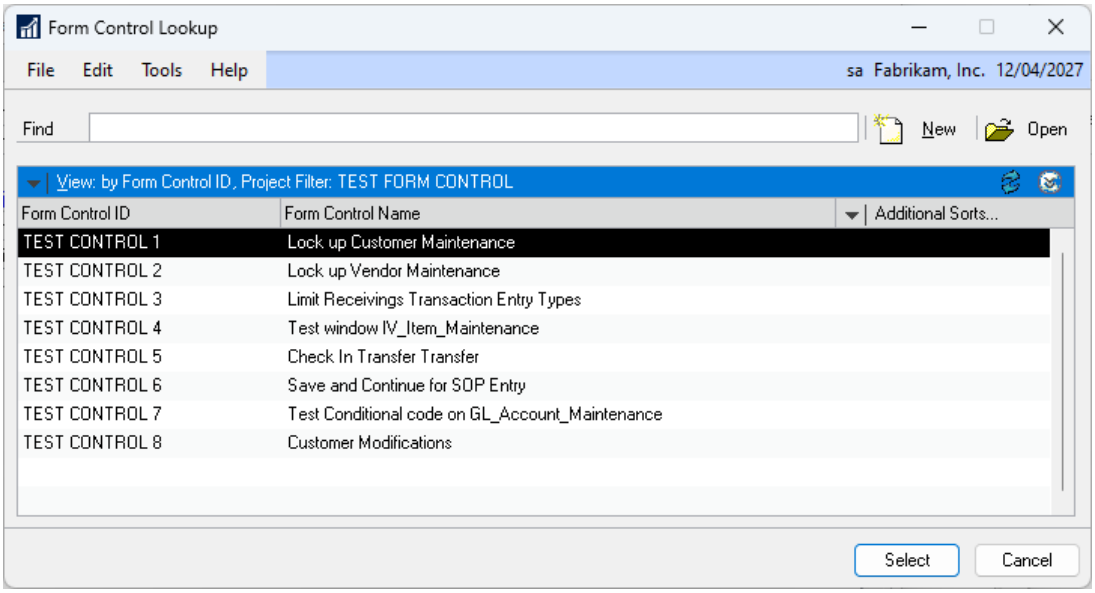



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

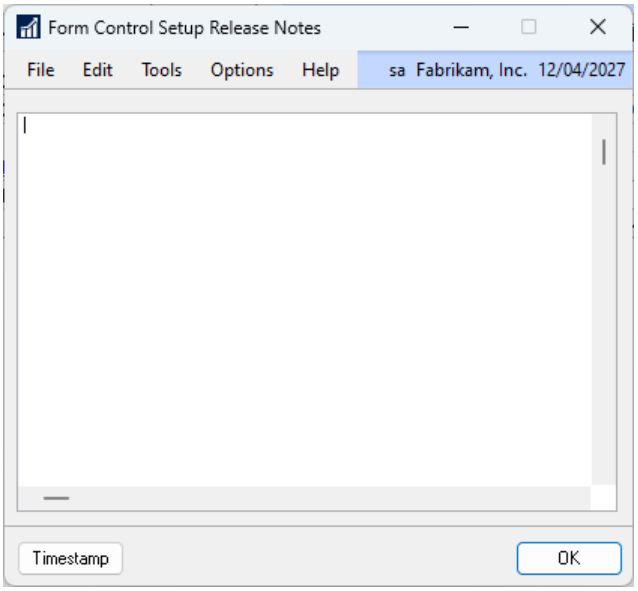
The following is a description of the individual fields on the window:

*Form Control ID*

This field contains a unique identifier for each Form Control in the system. The lookup button can be clicked to select from existing Form Control IDs. The lookup will be filtered to the current project if the Form Control belongs to a project.



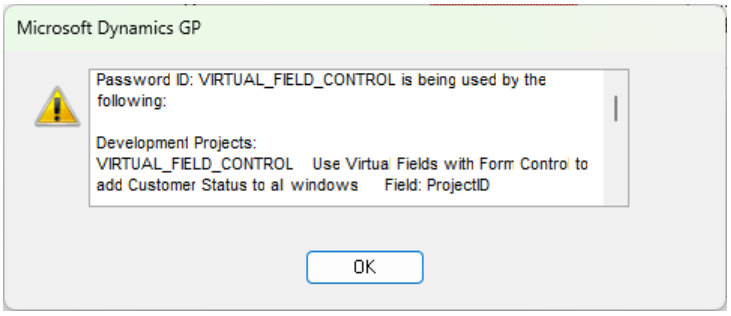

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Form Control IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

### *Form Control Information*

Click this button to display what resources are using this form control.

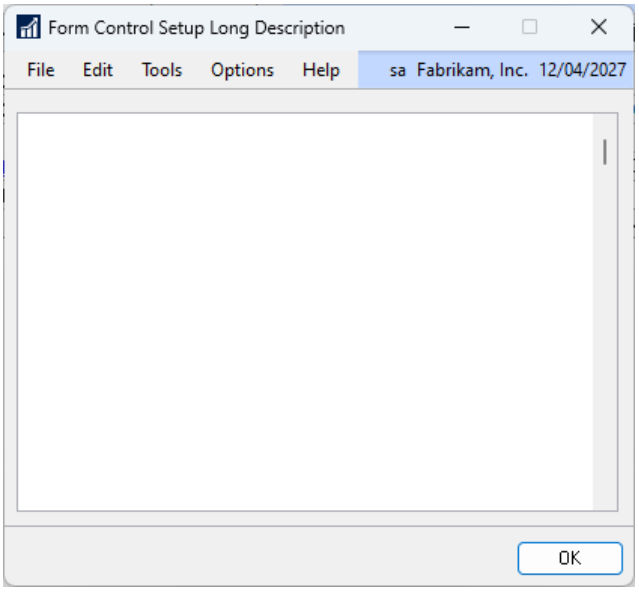



*Form Control Name*

This field contains a description of the form control.

*Long Description*

Use the Form Control Name expansion button to open the Long Description window. Use this field for a more detailed description of the form control.




*Disabled*

Use this checkbox to disable an entire form control ID.

*Manual Start Only*

Use this checkbox to prevent a Form Control ID starting automatically. It can be started manually with the MBS_Control_Start Helper Function.

*Minimize Log Entries*

This option can be enabled to prevent the form control generating entries in the GPPTools_<User>_<Company>.log file unless an error occurs.

*Project ID*

Use this field to add the current form control to a development project.

*Control Mode*

Use this drop down list to add rule templates to the current form control. There are templates for making Maintenance and Address windows read only and for blocking Save, Delete and Insert events.

*Enable in Web Client*

This checkbox tells GP Power Tools that this form control should be enabled for the Microsoft Dynamics GP Web Client .

*Enable in Service Mode*

This checkbox tells GP Power Tools that this form control should be enabled when Microsoft Dynamics GP is being executed in Service Mode (for Service Based Architecture) or when in Dynamics Process Server (DPS) mode.

*Form Control Tree*

This tree contains all the rules and the resource filters for each rule. Every form control ID will have a Base Settings rule and at least one Resource Filter which is used for the initial check if a form control ID should be used when a form is opened. Individual rules can be disabled and enabled on the tree.

The buttons to the right of the tree can be used to add, edit and delete rules and resource filters. It can also be used to reorder rules and resource filters. Double clicking on a rule or resource filter is the same as selecting the edit button.

See the sections below for more information on the Form Control Setup Rule window and the Form Control Setup Resource window.

The Add Button (plus symbol) has the following options available:

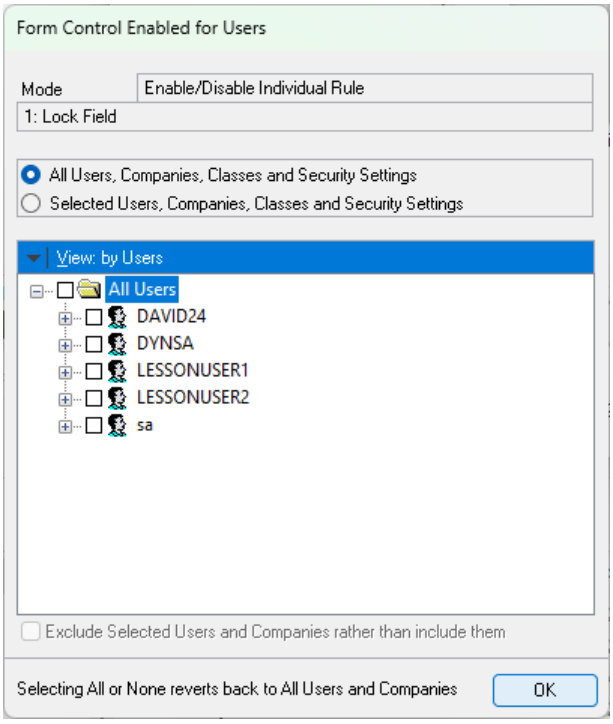
- Resource
- Form Rule
    - Password Form
    - Disable Form
    - Form Menu Shortcut
    - Reject Form Pre Script
    - After Form Pre Script
    - Reject Form Post Script
    - After Form Post Script

- Window Rule
    - Password Window
    - Disable Window
    - Change Window Title
    - Clear Changes Before Window Close
    - Focus First Window Field
    - Reject Window Pre Script
    - After Window Pre Script
    - Reject Window Post Script
    - After Window Post Script
    - Reject Window Activate Script
    - After Window Activate Script
- Scrolling Window Rule
    - Lock Scrolling Window
    - Reject Scrolling Window Save
    - After Scrolling Window Save
    - Reject Scrolling Window Delete
    - After Scrolling Window Delete
    - Reject Scrolling Window Insert
    - After Scrolling Window Insert
    - Reject Scrolling Window Fill
    - After Scrolling Window Fill
    - Reject Scrolling Window Pre
    - After Scrolling Window Pre
    - Reject Scrolling Window Post
    - After Scrolling Window Post
- Field Rule
    - Password Field Before
    - Password Field After
    - Warning Field Before
    - Lock Field
    - Disable Field
    - Hide Field
    - Default Field Value
    - Clear Field Value
    - Set Field Value
    - Strip Invalid Field Characters
    - Validate Field Value
    - Format String Field Value
    - Mask Field Value
    - Uppercase Field Value
    - Set Field Background Color
    - Set Field Font Color
    - Add Required Field
    - Enable Autocomplete Field
    - Round Decimals Field Value
    - Change Field Caption
    - Clear Changes Before Field
    - Set Focus to Field
    - Set Focus to Next Field
    - Reject Field Pre Script
    - After Field Pre Script
    - Reject Field Change Script
    - After Field Change Script
    - Reject Field Post Script
    - After Field Post Script

    - After Field Value Changed Script
- Label

See the Form Control Rule Types section above for more information about how each Rule type can be used.

Use the Rule Users button to specify which users and companies the selected form control rule should be active for. Once clicked Form Control Enabled for Users window will open in rule mode.




You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

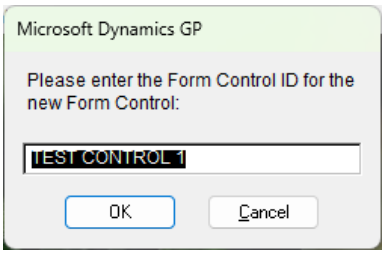
*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies, rather than include them option, allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the form control rule should not be active for.

The following is a description of the additional buttons on the window:

*Duplicate Button*

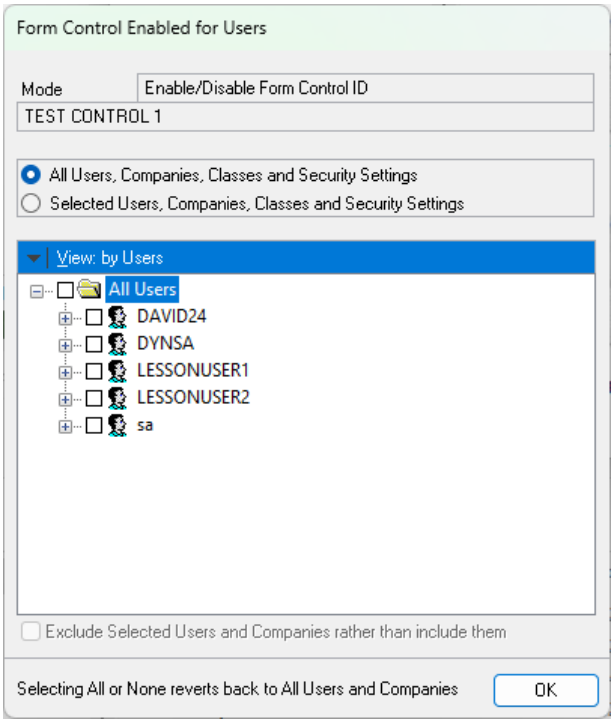
Use this button to duplicate or rename the current Form Control ID to a new Form Control ID. This is useful when an existing Form Control ID is very similar to the new one you want to create.




A new Form Control ID must be specified in the dialog which opens.

*Users Button*

Use this button to specify which users and companies the form control should be active for. Once clicked Form Control Enabled for Users window will open in form control mode.




You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the form control should not be active for.

*Status Button*

Use this button to open the Form Control Status window.

*Resources Button*

Use this button to open the Form Control Resources window.

The following is a description of the Options menu available for the window:

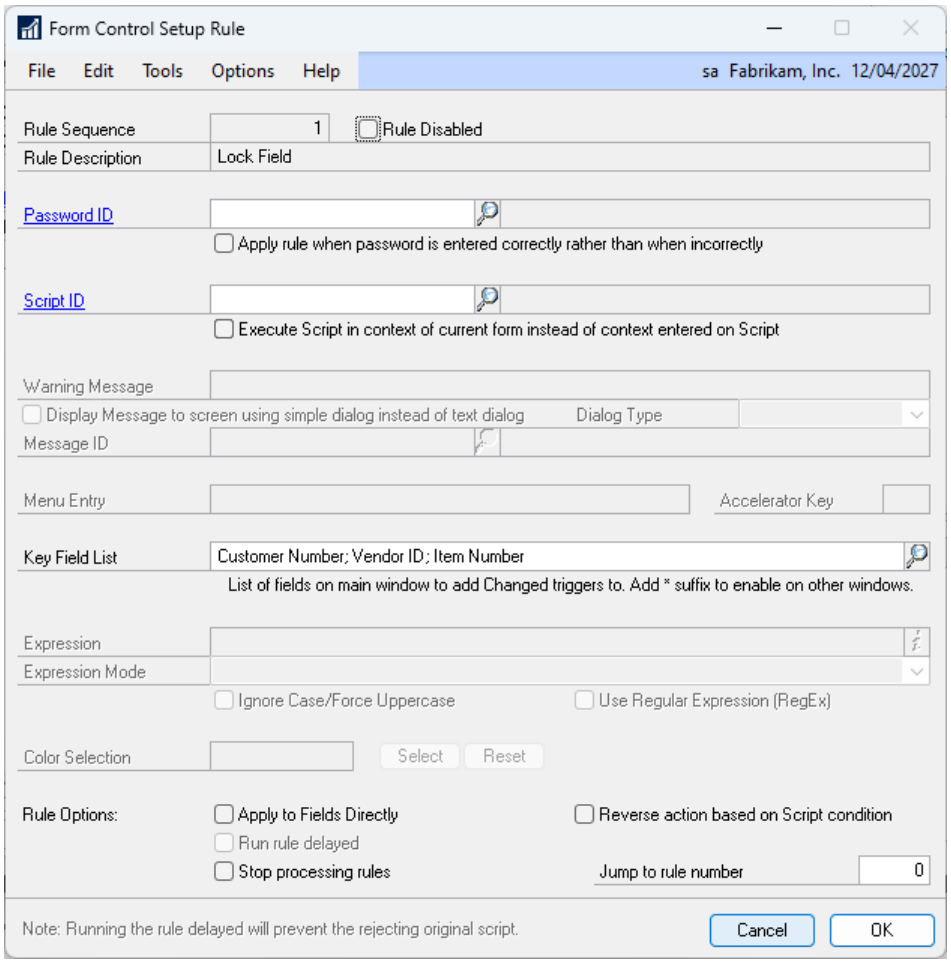
*Save and Continue*

Use this menu option to save the current form control without clearing the window. Control-S can be used as a shortcut.

## Form Control Setup Rule

The Form Control Setup Rule window is displayed when adding or editing a Rule. The fields enabled depend on the rule type selected. The Rule Sequence number and Rule Description are displayed at the top of the window.

*The Base Settings which exist for every Form Control ID have a Rule Sequence of zero and cannot be deleted or reordered.*




The following is a description of the individual fields on the window:

*Rule Disabled*

This checkbox can be used to disable the current rule. It is the same as using the checkbox on the Form Control Setup window.

*Password ID*

Enter an optional Password ID to be requested before processing the current rule.

*Apply rule when password its entered correctly*

Use this checkbox to reverse the behavior of the password entry success or failure.

*Script ID*

Enter an optional Script ID for a Runtime Execute Setup Form Control Conditional script using the 5261: Form.ControlConditional Script Purpose.

*Execute Script in context of current form*

Use this checkbox to override the dictionary context stored on the Form Control Conditional script with the dictionary context of the current form being processed by Form Control.

*Warning Message*

This field contains the message which will be displayed if the Form Control rule is not met. Depending on the rule type, the message can contain the %1 placeholder which will be substituted with the Expression field when the message is displayed.

The message can be programmatically overridden by using the MBS_Control_Update_Dialog Helper Function from the Form Control Conditional script.

*Display Message to screen using simple system dialog instead of text box dialog*

Select this checkbox if you want the message displayed to the screen in a simple system dialog instead of a text box dialog.

*Dialog/Alert Type*

Use this drop down list to select between Information, Warning (default), Error and Debug dialogs and desktop alerts. Debug dialogs and desktop alerts are only shown when the Debug menu is enabled, and Show Debug Messages is enabled. These Debug settings can be changed on the Dex.ini Settings window.

*Message ID*

Use this field to define a Message ID to be used instead of the default Warning Message. Messages have the advantage of only being defined once and can automatically change depending on the language of the system. To setup Messages use the Messages Setup window.

*Menu Entry*

This field contains the description to be displayed on the Form Menu created by this form control.

*Accelerator Key*

This field contains an optional accelerator shortcut key (used with Control) for the menu entry.

*Key Field List*

This field contains a list of fields which are used in different ways depending on the current Rule type. Instructions for how the Key Field List should be used will be displayed on the line under the field. If including more than one field, use a semicolon (;) as a separator.

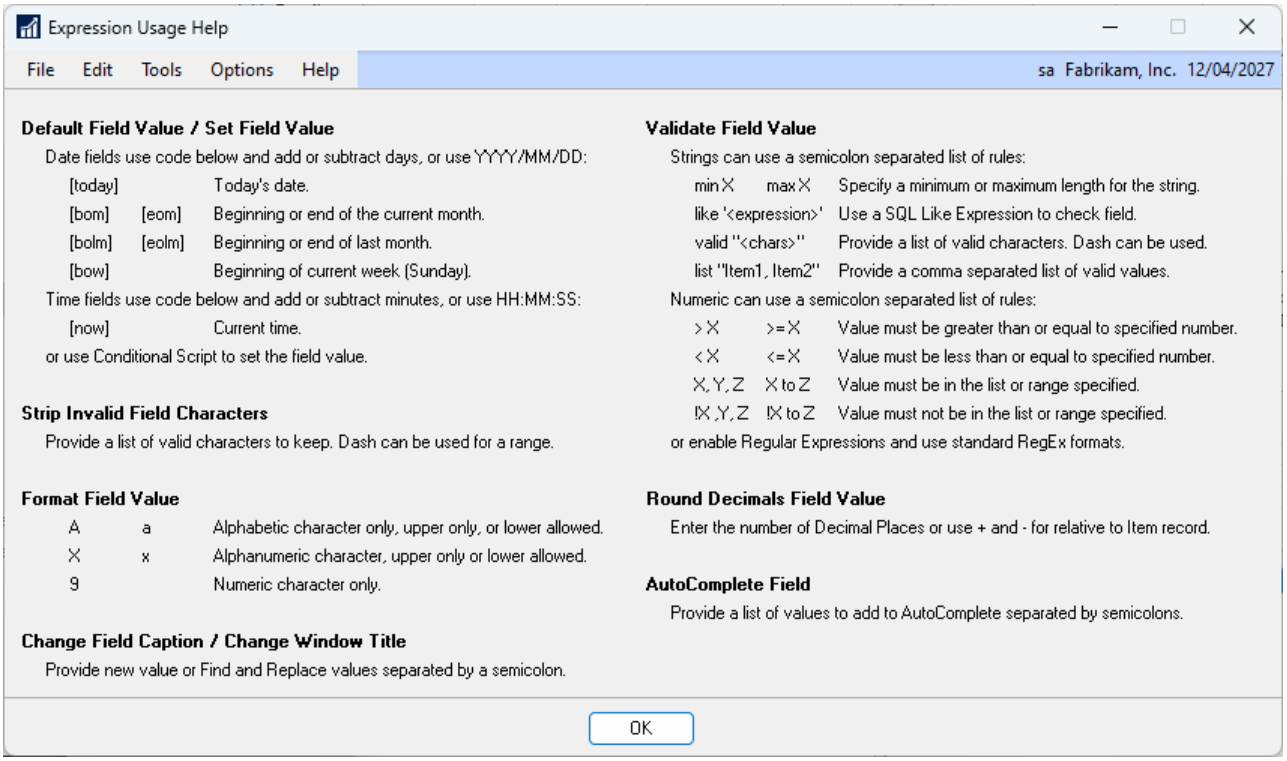
### *Expression*

This field contains a list of expressions which are used in different ways depending on the current Rule type. Information on how the Expression field can be used can be viewed by clicking the Expression Information Button. If including more than one expression, use a semicolon (;) as a separator.

The expression can be programmatically overridden by using the MBS_Control_Update_Expression Helper Function from the Form Control Conditional script.

### *Expression Usage Help Button*

Click this button to open the Expression Usage Help window.




### *Expression Mode*

This drop down list is used with the Set Decimal Field rule to specify if basing the Decimal places relative to the Currency or Quantity Decimal Places defined on the Item Master table.

### *Ignore Case/Force Uppercase*

This checkbox is used to specify that the expression should be treated as case insensitive, or the resulting data should be forced to uppercase depending on the current Rule type.

### *Use Regular Expression (RegEx)*

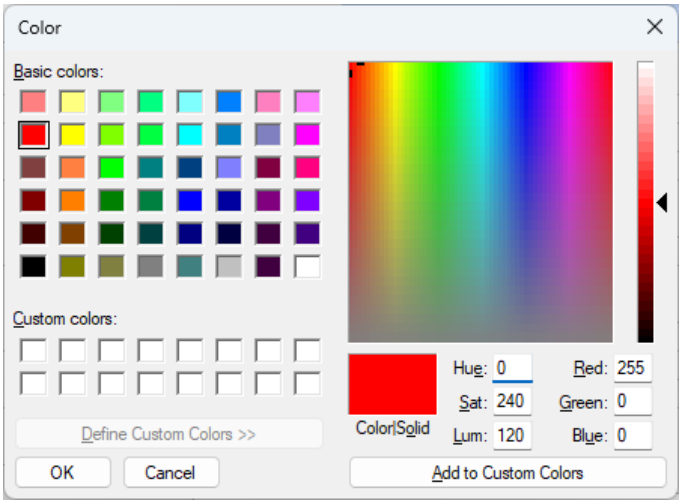
This checkbox is used to specify that the Expression for the Validate Field Value rule should be interpreted as a Regular Expression rather than the simpler GP Power Tools validation rules. Information on how to use Regular Expressions (RegEx) is available via an internet search.

### *Color Selection*

This read only field displays the current color stored for the Set Field Background Color or Set Field Font Color Rule types.

### *Select Button*

Click this button to open the color selection dialog. Click the Define Custom Colors to see the full color spectrum and select from 16.7 million available 24-bit colors.




### *Reset Button*

Click this button to reset back to the default system color for the background or font.

### *Apply to Fields Directly*

Use this checkbox to apply triggers directly to the fields included the Resource Filter in addition to referencing them from Window level events. This can be used if other code on a window is preventing form control rules from working.

### *Run rule delayed*

Use this checkbox to change the timing of when the rule is applied. Running delayed will ensure the rule is applied after any existing code has completed.

*Running a "Reject" rule type using run delayed will mean that the rule will be queued to run before the original code, but it will run after all other code has completed. This will prevent it being able to reject the original script.*

### *Reverse action based on Script condition*

This checkbox can be used to alter the behavior of a reversable rule. Normally the Form Control Conditional script will decide if the rule is applied or not applied. With a reversable rule, this checkbox will change the behavior to either apply or un-apply the rule.

*For example: With the Lock Field rule, if the checkbox is not selected, the script can decide if the field is locked or left unchanged. If the checkbox is selected, the script can decide if the field locked or unlocked.*

*Stop processing rules*

Select this checkbox to stop processing any subsequent rules for the current Form Control ID after processing the current rule. It will not take effect if the Form Control Conditional script decides the rule should not be processed.

*Jump to rule number*

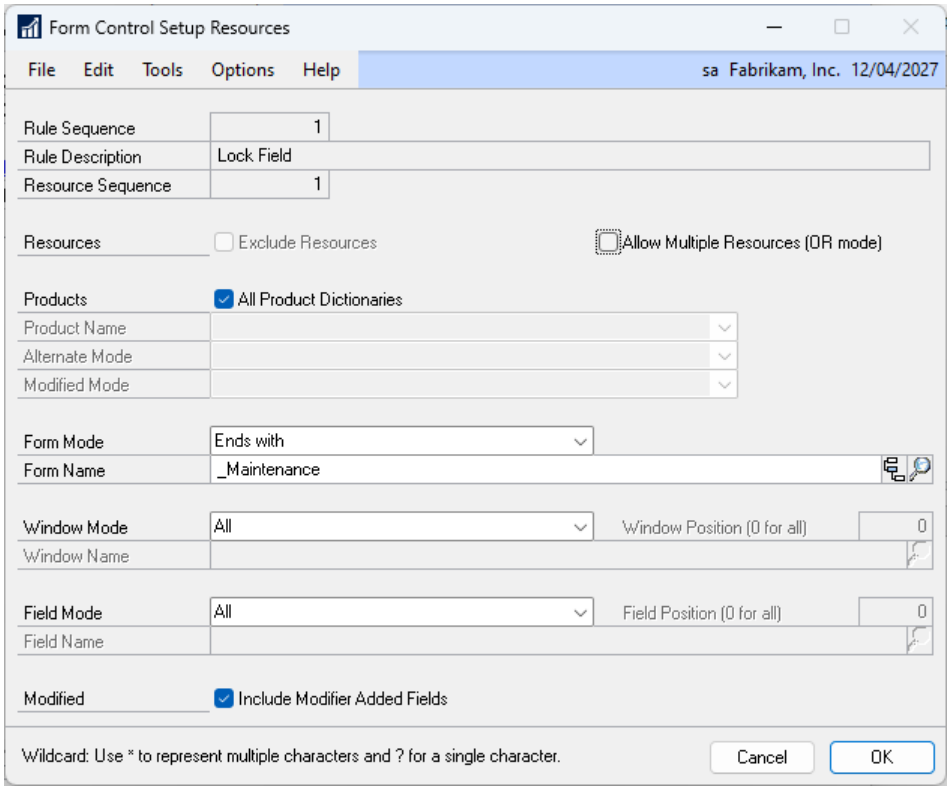
Put a rule number into this field to jump to a rule other than the next rule in the sequence for the Form Control ID. This option can only jump to rule numbers higher than the current rule. It will not take effect if the Form Control Conditional script decides the rule should not be processed.

## Form Control Setup Resources

The Form Control Setup Resources window is displayed when adding or editing a Resource Filter. The fields enabled depend on the rule type selected. The Rule Sequence number, Rule Description and Resource Sequence are displayed at the top of the window.

*The Base Settings which exist for every Form Control ID have a Rule Sequence of zero and cannot be reordered. Window and Field settings are not available for the Base Settings Resource Filter.*

*The first Resource Filter for each rule has a Resource Sequence of 1 and cannot be deleted or reordered and cannot be marked to Exclude Resources.*




The following is a description of the individual fields on the window:

*Exclude Resources*

Select this checkbox to exclude any resources defined in the window from the Resource Filter.

*Allow Multiple Resources (OR mode)*

Select this checkbox to allow more than one include Resource selections. It must be selected for all the included resources for the current rule which should be treated with OR logic.

*All Product Dictionaries*

Select this checkbox to select all product dictionaries for the filter.

*Product Name*

This drop down list is used to select a specific product dictionary when All Product Dictionaries is unselected.

*Alternate Mode*

This field is used to select whether the current Form Control ID should be enabled for both original and all alternate forms, or if it should only be enabled when the form is in a specific product dictionary. Use dictionary 0: Microsoft Dynamics GP for only the original form or select the product dictionary which contains the desired alternate form. This option is only available for the Base Settings Resource Filter.

*Modified Mode*

This field is used to select whether the current Form Control ID should be enabled for both original and modified forms, or if it should only be enabled for original forms or for modified forms. This option is only available for the Base Settings Resource Filter.

*Form Mode*

Use this drop down to select the search mode for Forms. The options are All, Begins with, End with, Contains, Exact, Wildcard and None. Once the search mode is selected, use the Form Name field to add the search term.

*Form Name*

Enter the search term to use in conjunction with the Form Mode to specify the search filter. For Wildcard mode use an asterisk (*) to represent multiple characters and a question mark (?) to represent a single character.

*Window Mode*

Use this drop down to select the search mode for Windows. The options are All, Begins with, End with, Contains, Exact, Wildcard and None. Once the search mode is selected, use the Window Name field to add the search term.

*Window Position*

This field can be used to limit the windows with a form that the Resource Filter is applied to. Leave as zero for all. Use 1 to limit to the first or main window on a form.

*Window Name*

Enter the search term to use in conjunction with the Window Mode to specify the search filter. For Wildcard mode use an asterisk (*) to represent multiple characters and a question mark (?) to represent a single character.

*Field Mode*

Use this drop down to select the search mode for Fields. The options are All, Begins with, End with, Contains, Exact, Wildcard and None. Once the search mode is selected, use the Field Name field to add the search term.

*Field Position*

This field can be used to limit the fields with a window that the Resource Filter is applied to. Leave as zero for all.

*Field Name*

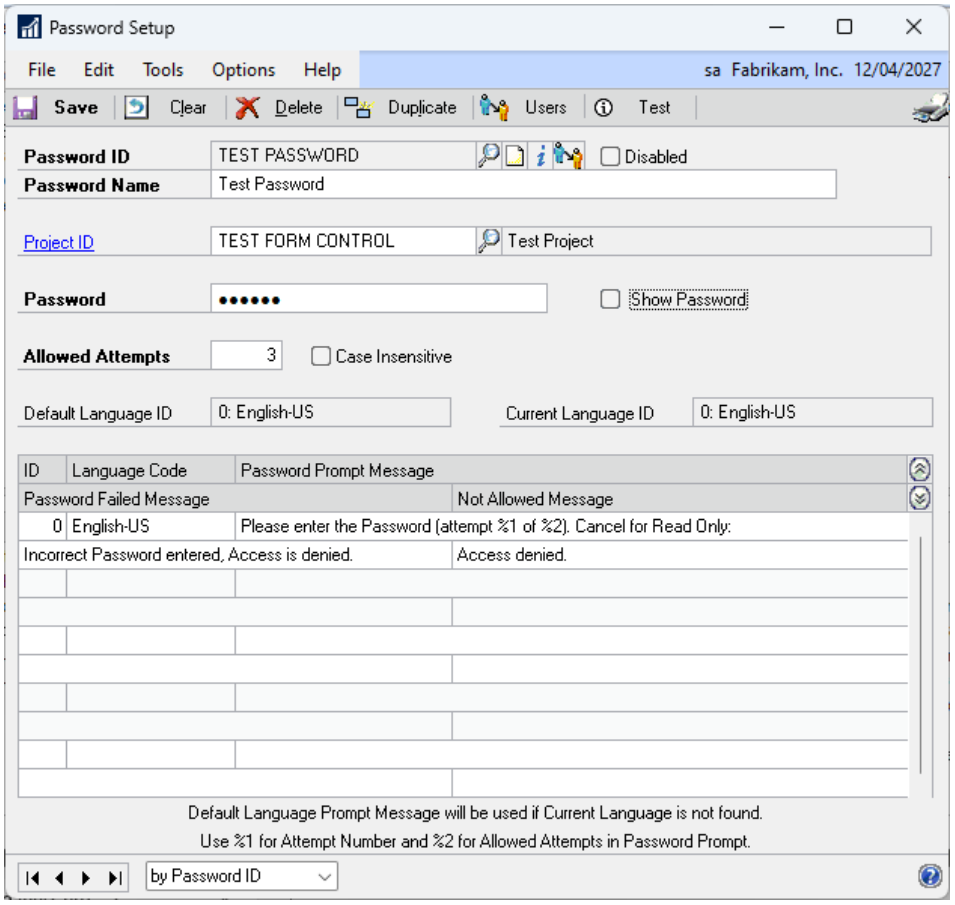
Enter the search term to use in conjunction with the Field Mode to specify the search filter. For Wildcard mode use an asterisk (*) to represent multiple characters and a question mark (?) to represent a single character.

*Include Modifier Added Fields*

Select this checkbox if fields added with Modifier to modified windows should be included within the Resource Filter.

## Password Setup

You can open the Password Setup window by selecting Password Setup from the Cards section of the GP Power Tools Area Page or by selecting Password Setup from the Options button drop list on the main window. This is an Advanced Mode feature.

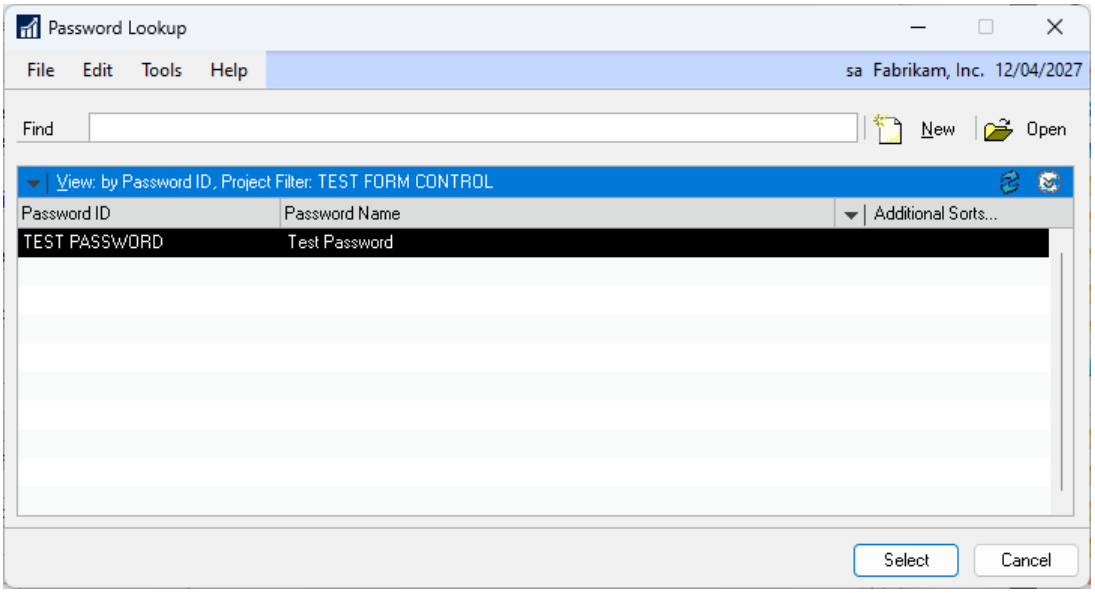



When using the browse buttons on this window when the sort by drop down list is set to by Project ID, there will be a soft limit at the first and last record within a project. Clicking the browse button a second time will move past the soft limit.

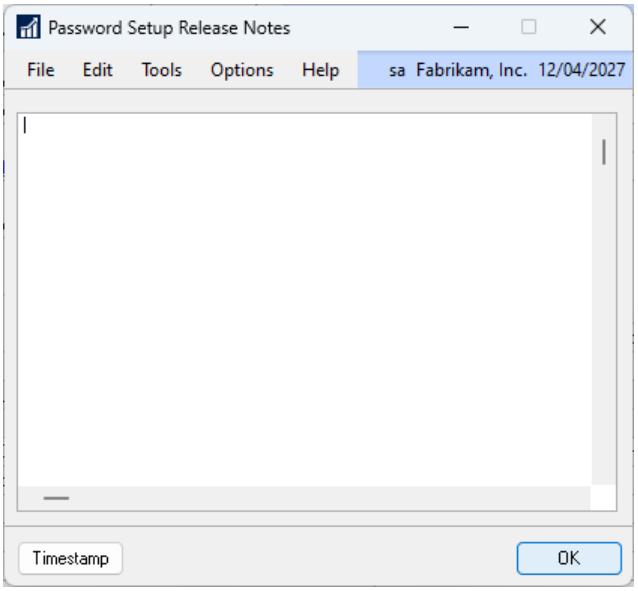
The following is a description of the individual fields on the window:

*Password ID*

This field contains a unique identifier for each Password in the system. The lookup button can be clicked to select from existing password IDs. The lookup will be filtered to the current project if the password belongs to a project.



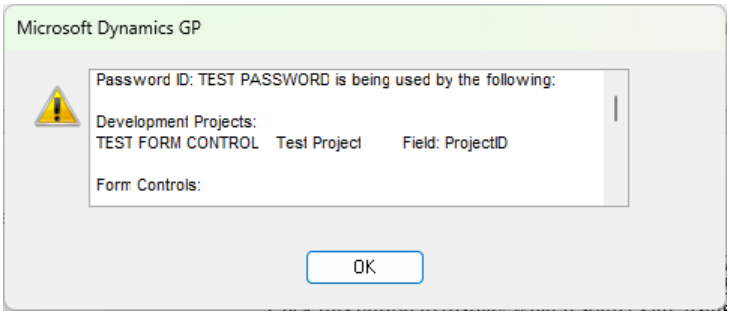

The Notes Button can be clicked to enter Release Notes. Use the Timestamp Button to add a timestamp to the bottom of the release notes.




*Note that the Password IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

### *Password Information*

Click this button to display what resources are using this password.




*Password Name*
This field contains a description of the password.

*Disabled*
Use this checkbox to disable the password. If disabled, the password request will be successful without opening any dialogs.

*Project ID*
Use this field to add the current password to a development project.

*Password*
Enter the password to be used in this field. By default, it will not be displayed.

*Show Password*
Use this checkbox to show the password characters being entered.

*Allowed Attempts*
Specify how many attempts to enter the password are allowed before being rejected.

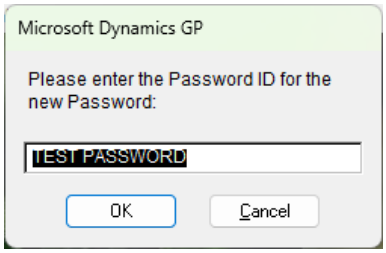
*Case Insensitive*
Use this checkbox to specify that case for the password characters is not important, if desired.

*Prompt List*
This scrolling window is where you can enter the prompts and messages used for each language. If there are no prompts for a particular language, the prompts for the default language will be used instead.

The following is a description of the additional buttons on the window:

### *Duplicate Button*

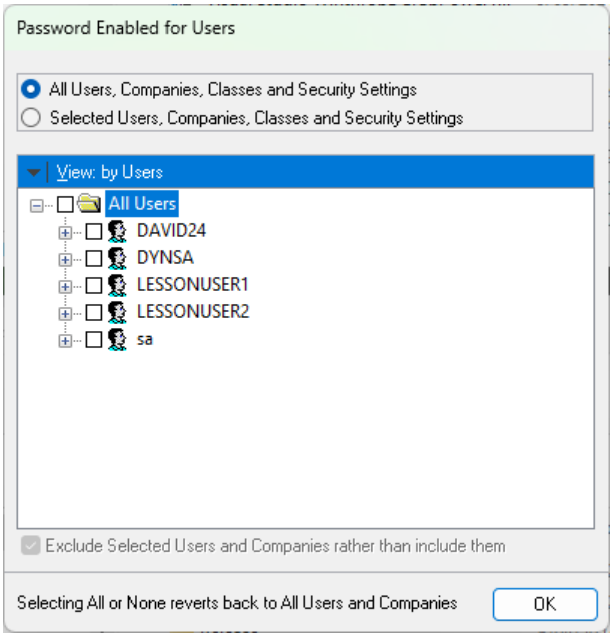
Use this button to duplicate or rename the current Password ID to a new Password ID. This is useful when an existing password ID is very similar to the new one you want to create.




A new password ID must be specified in the dialog which opens.

### *Users Button*

Use this button to specify which users and companies the password should be active for. Once clicked Password Enabled for Users window will open.



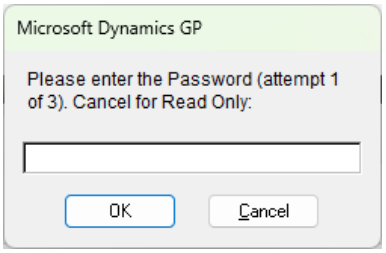

You can view this window by users or by companies and navigate the tree to select the user and company combinations as required. You can also select by User Classes, Security Roles, Security Tasks and Security Modified Alternate IDs.

*If all users are selected on the tree, the tree selections will be cleared and the mode will change from Selected Users and Companies to All Users and Companies. If no users are selected on the tree, the mode will change to All Users and Companies. A red visual cue will be displayed if there are user settings present.*

The Exclude Selected Users and Companies rather than include them option allows you to invert the behavior of the window. This is handy when it is easier to specify the users and companies for whom the password should not be active for.

### *Test Button*

Use this button to display a test of the password dialog and display the results afterwards.




The following is a description of the Options menu available for the window:

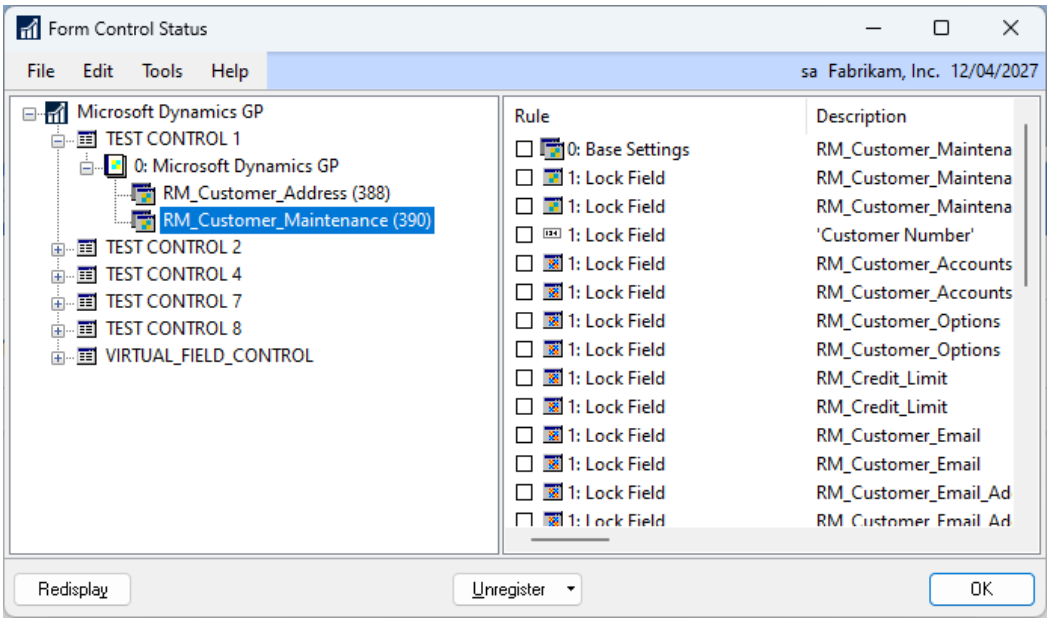
### *Save and Continue*

Use this menu option to save the current password without clearing the window. Control-S can be used as a shortcut.

## Form Control Status

You can open the Form Control Status window by selecting Form Control Status from the Inquiry section of the GP Power Tools Area Page or by selecting Form Control Status from the Options button drop list on the main window. This is an Advanced Mode feature.

The Form Control Status window is used to see what triggers have been registered to implement the form control functionality.




The following is a description of the individual fields on the window:

*Form Control Tree*

This tree shows the currently active Form Control IDs and which forms in which dictionaries they are active for.

*Trigger List*

This list shows the form control triggers registered for the selected Form Control ID and dictionary form.

The following is a description of the additional buttons on the window:

*Redisplay Button*

Use this button to redisplay the data on the window. The data in the window is also redisplayed automatically when a new form or Form Control ID is added.

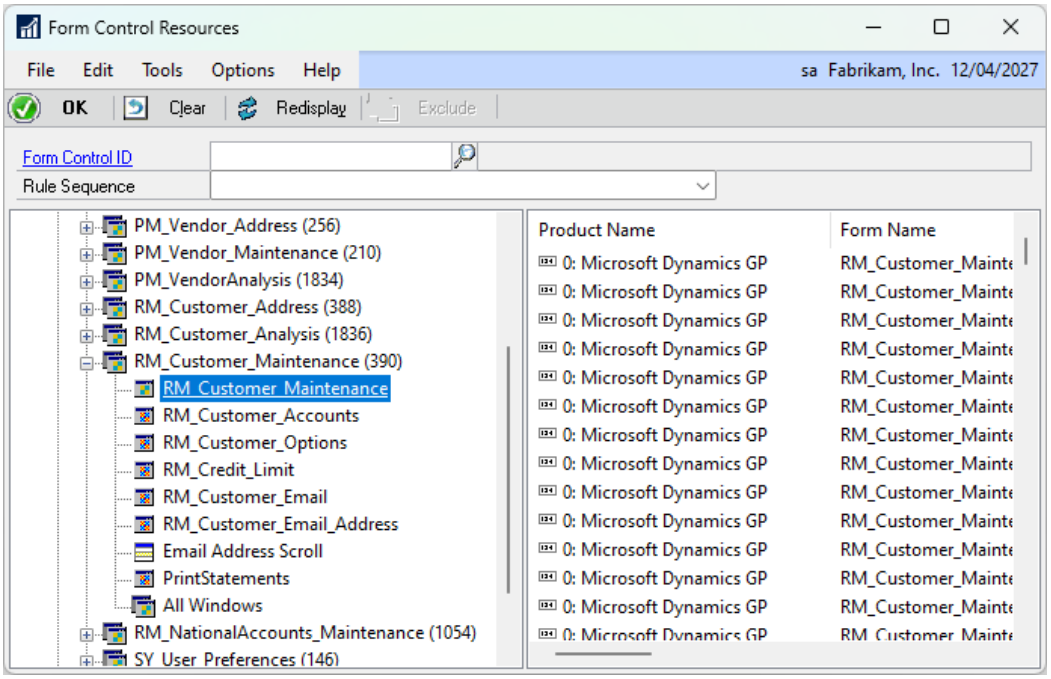
*Unregister Button*

Use this button to Unregister triggers for a Form Control ID and form, for all forms for a Form Control ID or for all Form Control IDs. Unregistered triggers will be registered again next time the form is opened.

## Form Control Resources

You can open the Form Control Resources window by selecting Form Control Resources from the Inquiry section of the GP Power Tools Area Page or by selecting Form Control Resources from the Options button drop list on the main window. This is an Advanced Mode feature.

The Form Control Resources window is used to show the form, window and field resources cached by form control for the current user and company. Entering a Form Control ID and Rule Sequence number will filter the displayed data to only show the resources that meet the resource filtering settings for the selected Form Control ID and rule.




The following is a description of the individual fields on the window:

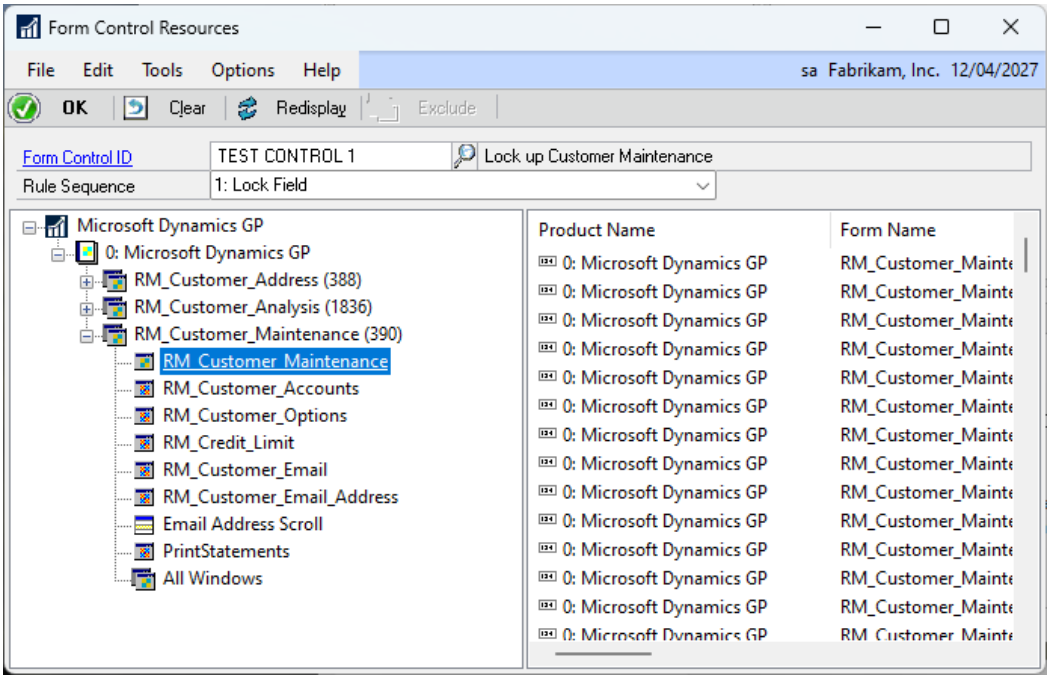
*Form Control ID*

This field can be used to filter the resources in the Form Control Resource Tree and Form Control Resource List to only show resources that meet the resource filter for the selected Form Control ID and Rule.

*Rule Sequence*

This field can be used to filter the resources in the Form Control Resource Tree and Form Control Resource List to only show resources that meet the resource filter for the selected Form Control ID and Rule.

When the filter is applied, the contents of the Resource Tree and Resource List are limited to only those resources that meet the filters for the selected Form Control ID and rule.




*Form Control Resource Tree*

This tree displays the dictionary, form and window information for the cached forms for the current user and company.

*Form Control Resource List*

This list displays the form, window and field information for the cached forms for the current user and company. Fields added by Modified or Alternate windows will use an icon with a blue or red pencil overlay (respectively).

The following is a description of the additional buttons on the window:

*Clear Button*

Use this button to clear the Form Control ID and Rule Sequence filter and redisplay the unfiltered resources.

*Redisplay Button*

Use this button to redisplay the resources on the window. The resources in the window are also redisplayed automatically when a new form is added.

*Exclude Button*

Use this button exclude the currently selected resource from the Form Control ID and Rule. This button is only available when a Form Control ID and Rule have been entered to filter the resources displayed.

The following is a description of the Options menu available for the window:

*Reset Resource Data*

Use this menu option to remove all the cached resources for the current user and company. This will close all open windows and clear the cache tables. Forms will be added back to the cache as they are next opened.

*The resources stored in the cache tables MBS_WindowControl_Form_TEMP (WPT50501) and MBS_WindowControl_Resource_TEMP (WPT50500) are automatically maintained by GP Power Tools. If the dictionary version changes, or the data has not been used for 30 days, the resources are removed and will be re-read on demand. Modified resources are removed when exiting Microsoft Dynamics GP to force them to be re-read on demand each session in case they have changed.*

# Chapter 7: Database Tools Features

This chapter includes the following sections:

- *XML Table Export**
- *XML Table Import**
- *Database Validation**
- *SQL Login Maintenance**
- *Password Reset Email Settings**
- *Copy User Settings**
- *SQL Trigger Control**
- *Note Fix Utility**
- *Database Space Recovery**
- *Additional Database Features*

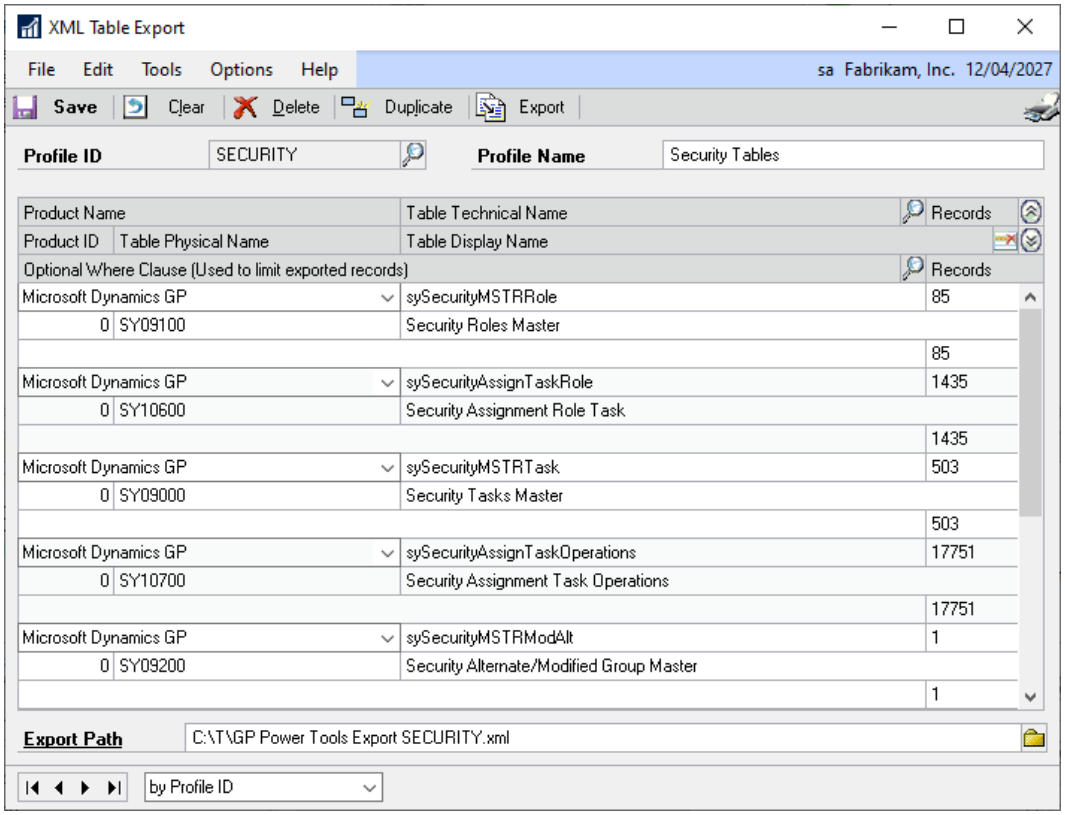
* *Advanced Mode Feature*

## XML Table Export

You can open the XML Table Export window by selecting XML Table Import from the Utilities section of the GP Power Tools Area Page or by selecting Database >> XML Table Export from the Options button drop list on the main window. This is an Advanced Mode feature.

The XML Table Export window can be used to copy the contents of one or more tables residing in any product into an XML file. All tables selected will be exported into the single XML file listed on the Export Path.

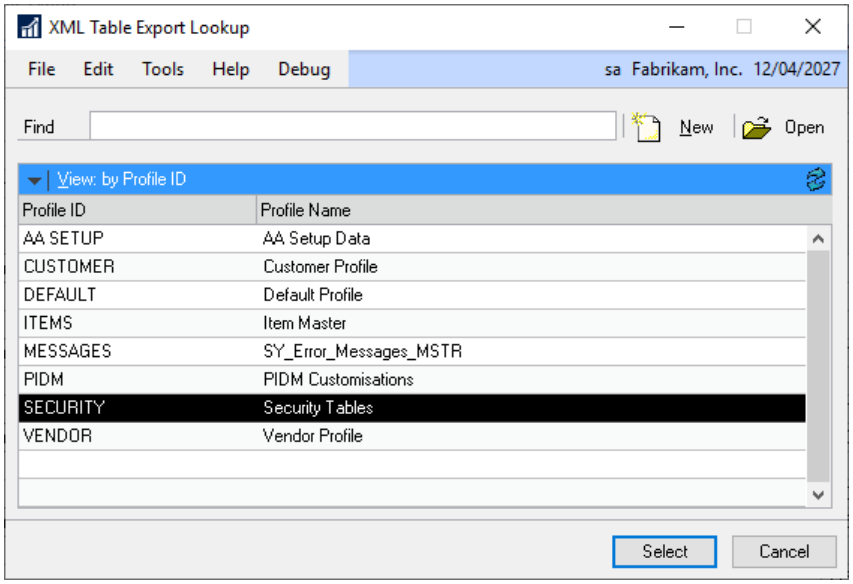
Using separate Profile IDs allows multiple sets of tables to be stored for related groups.




The following is a description of the individual fields on the window:

*Profile ID*

This field contains a unique identifier for each XML Table Export profile in the system. The lookup button can be clicked to select from existing profile IDs.




*Note that the Profile IDs starting with the prefix character of tilde (~) are reserved for use by Microsoft Support.*

### *Profile Name*

This field contains a description for the XML Table Export profile.

### *Table List*

Select the tables you want to export and add them to the list. You can use the lookup or manually enter the Table Technical Name or Table Physical Name fields.

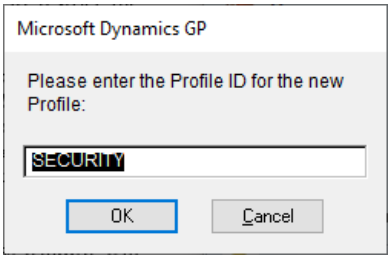
### *Export Path*

This field contains the path of the file name to which the tables will be export as XML.

The following is a description of the additional buttons on the window:

### *Duplicate Button*

Use this button to duplicate the current profile ID to a new profile ID. This is useful when an existing profile ID is very similar to the new one you want to create.




A new profile ID must be specified in the dialog which opens.

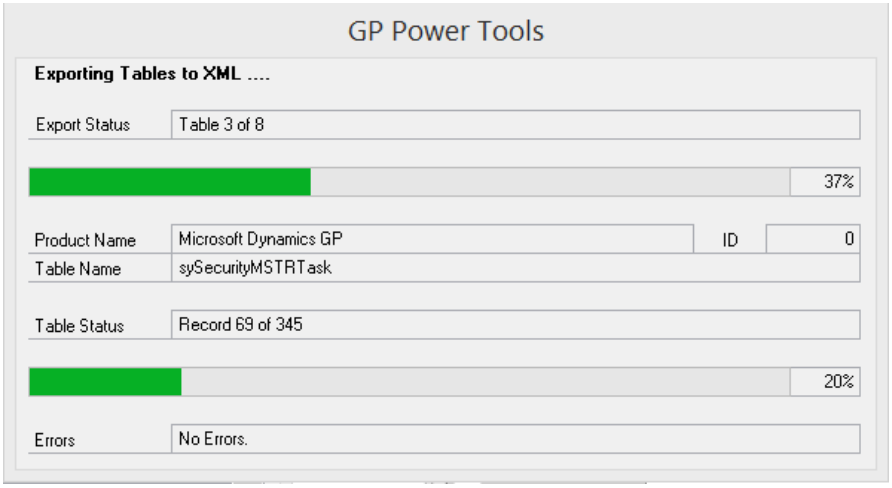
### *Export Button*

Use this button to export the data to the file named in the Export Path field.

For each table specified in the scrolling window section of this window, you can specify an Optional SQL Where Clause Clause to restrict the records export for that table.

*XML Table Export can be used to obtain a customer's data for specific tables without requiring a full SQL database backup. Just select the tables for which you need the data and click OK to save the selection. Then use the Configuration Export/Import window to export the setting file to send to the customer. The customer can then import the settings and use the XML Export window to export the desired tables.*

During the export or import process, the following progress window will be displayed.

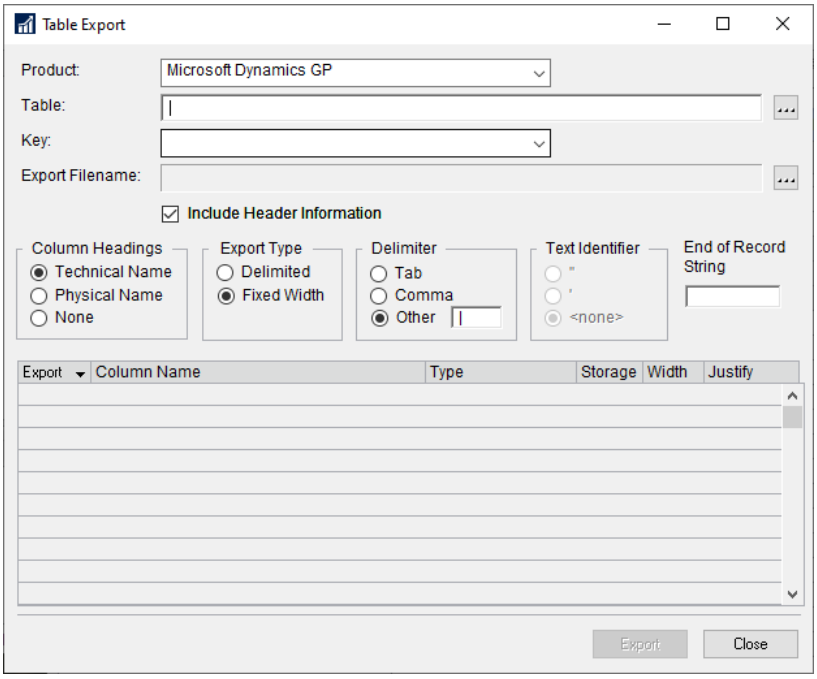



*XML Table Export can be used to backup data before running test scenarios so the data can be restored afterwards to allow the scenarios to be run again with the same initial data.*

*There is no data validation or business logic checking when data is imported using XML Table Import. This is similar to the Dexterity Table Import Utility. It is best to ensure that all related tables are exported by XML Table Export.*

*If both the XML Table Export window and XML Table Import window are open, the import path will default to the export path from the XML Table Export window.*

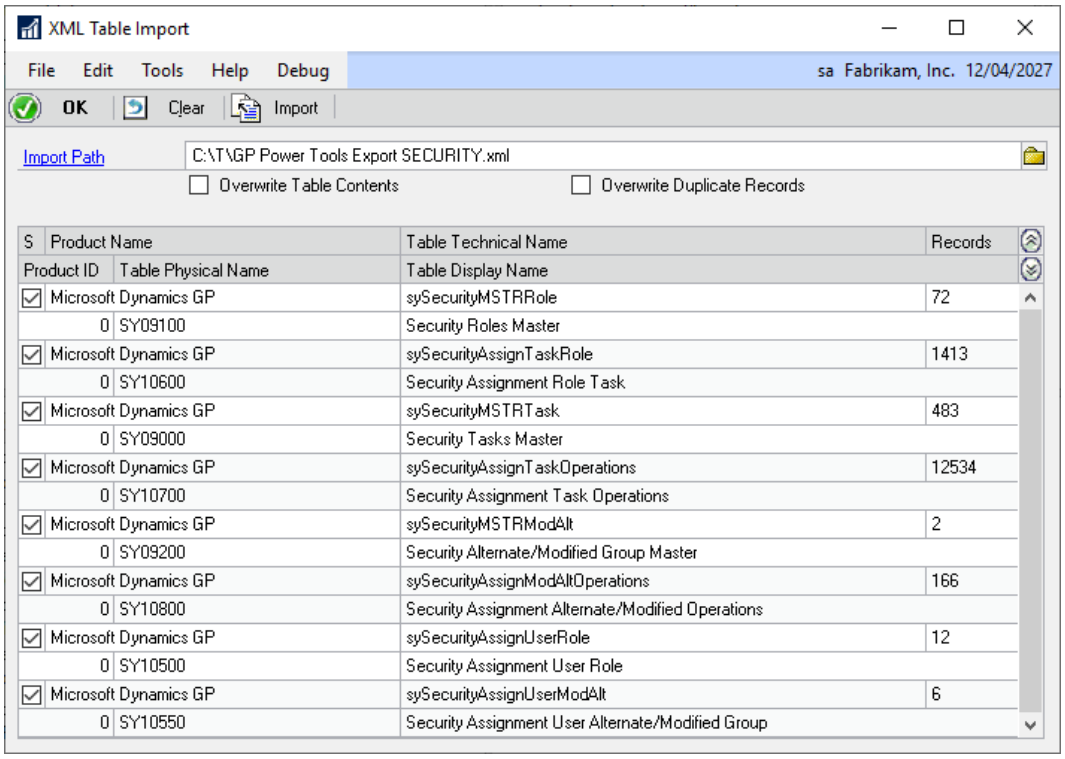
From the Options menu, you can open the built in Table Export tool if you want to export data as a text file.



## XML Table Import

You can open the XML Table Import window by selecting XML Table Import from the Utilities section of the GP Power Tools Area Page or by selecting Database >> XML Table Import from the Options button drop list on the main window. This is an Advanced Mode feature.

The XML Table Import window can be used to import the contents of a number of tables from an XML file previously exported by the XML Table Export window.




| S | Product Name | Table Technical Name | Records |
|---|---|---|---|
| Product ID | Table Physical Name | Table Display Name | |
| ☑ | Microsoft Dynamics GP | sySecurityMSTRRole | 72 |
| 0 | SY09100 | Security Roles Master | |
| ☑ | Microsoft Dynamics GP | sySecurityAssignTaskRole | 1413 |
| 0 | SY10600 | Security Assignment Role Task | |
| ☑ | Microsoft Dynamics GP | sySecurityMSTRTask | 483 |
| 0 | SY09000 | Security Tasks Master | |
| ☑ | Microsoft Dynamics GP | sySecurityAssignTaskOperations | 12534 |
| 0 | SY10700 | Security Assignment Task Operations | |
| ☑ | Microsoft Dynamics GP | sySecurityMSTRModAlt | 2 |
| 0 | SY09200 | Security Alternate/Modified Group Master | |
| ☑ | Microsoft Dynamics GP | sySecurityAssignModAltOperations | 166 |
| 0 | SY10800 | Security Assignment Alternate/Modified Operations | |
| ☑ | Microsoft Dynamics GP | sySecurityAssignUserRole | 12 |
| 0 | SY10500 | Security Assignment User Role | |
| ☑ | Microsoft Dynamics GP | sySecurityAssignUserModAlt | 6 |
| 0 | SY10550 | Security Assignment User Alternate/Modified Group | |

Select the XML file as the Import Path. The tables contained in the file will be listed.

Select the tables you want to import and then click Import to start importing.

*When importing data into tables it is possible that the tables already contain data and that duplicate records may occur. XML Table Import has overwrite options to handle this situation.*

The following Overwrite options are available:

*Overwrite Table Contents*

Checking this option will cause the original contents of the table to be deleted prior to importing the XML file. None of the original data will be kept.

*Overwrite Duplicate Records*

Checking this option will allow XML Table Import to overwrite a duplicate record with the data from the XML file. If this option is not checked and a duplicate occurs, the data from the XML file will be ignored and a duplicate record error logged.

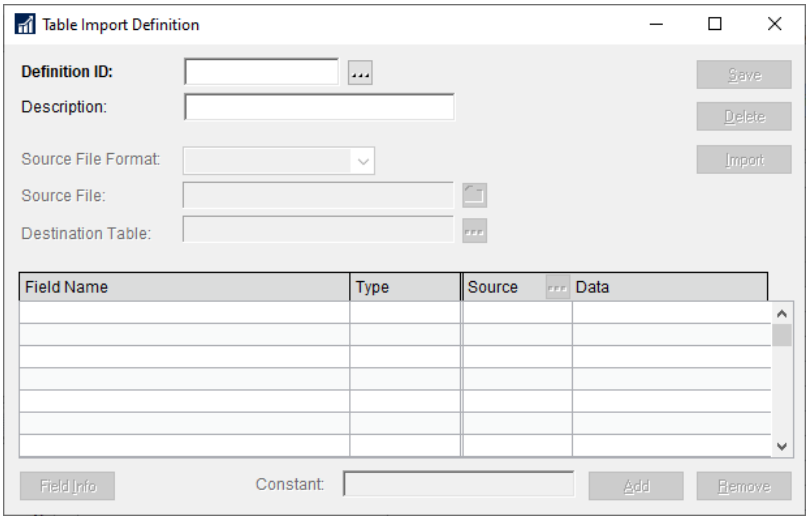
During the export or import process, the progress window will be displayed.

*XML Table Import can be used to restore data from backups you made before running test scenarios. This allows the scenarios to be run again with the same initial data.*

*There is no data validation or business logic checking when data is imported using XML Table Import. This is similar to the Dexterity Table Import Utility. It is best to ensure that all related tables are exported by XML Table Export.*

*If both the XML Table Export window and XML Table Import window are open, the import path will default to the export path from the XML Table Export window.*

From the Options menu, you can open the built in Table Import tool if you want to import data from a text file, without any validation or business logic.



## Database Validation

Please make sure you review the Using Database Validation section for the steps to use this window.

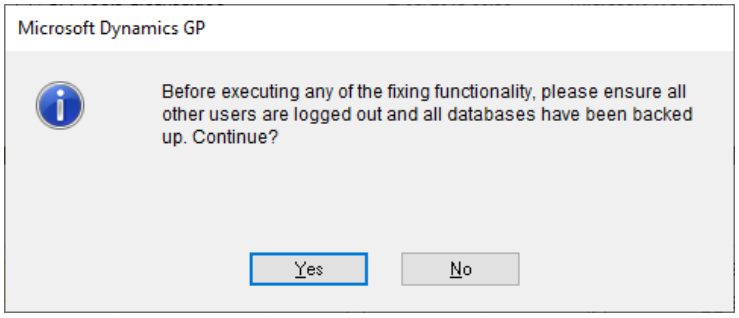
You can open the Database Validation window by selecting Database Validation from the Utilities section of the GP Power Tools Area Page or by selecting Database >> Database Validation from the Options button drop list on the main window. This is an Advanced Mode feature.

The Database Validation window is designed to perform a number of system checks to ensure that your SQL Server settings and databases correctly match what is expected by Microsoft Dynamics GP. If any issues are found, the Database Validation will provide options to resolve them.

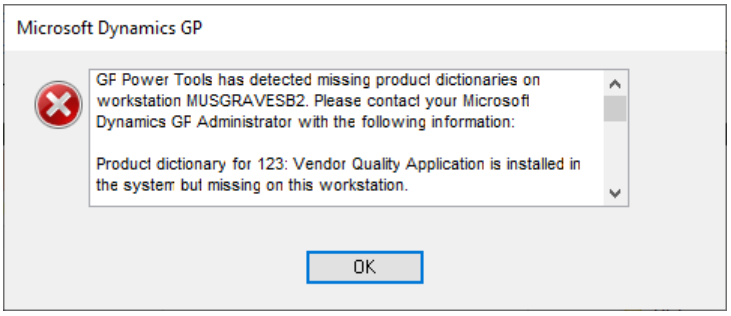
Running Database Validation before upgrading or after copying databases between SQL Servers can resolve any potential issues before they occur.

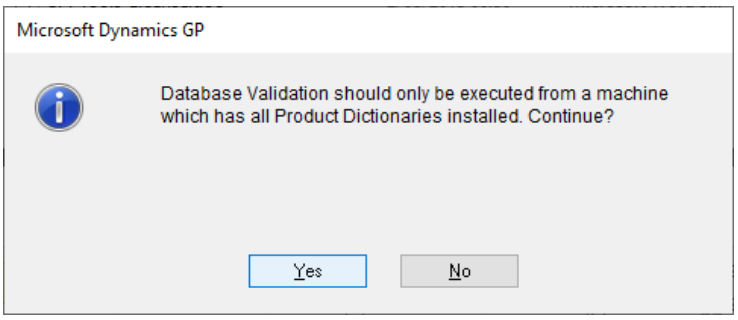
Before the window opens you will be reminded to ensure that all users are logged out and that your company and system databases have been backed up before executing any of the fixing functionality.

*Running the Database Validation checks to identify issues is read-only and does not require backups or exclusive use of the system.*




Before the window opens the system will check if you have the dictionaries for all products installed. If not, the following dialogs will be displayed with the details of the missing products. This is to ensure that all dictionaries are present when comparing SQL Server tables to tables in the dictionaries.






When the window first opens, Database Validation performs its first series of checks.

**Users and Databases**:

- Confirm that the 'DYNSA' SQL Server Login exists
- Confirm that 'DYNSA' is assigned as dbo for the SQL Databases
- Confirm that the 'DYNSA' GP User ID exists
- Confirm that 'DYNSA' is assigned access to all GP Companies
- Confirm that 'sa' is assigned access to all GP Companies
- Identify GP Companies for which there is no SQL Database
- Identify GP User IDs for which there is no SQL Login
- Identify GP Users for not assigned to the DYNGRP SQL Role
- Identify Company Access records for missing Users or Companies
- Identify missing Database Users as per Company Access records

**Dynamics GP Utilities**:

Using the records in DB_Upgrade and DU00020 tables in the System database.

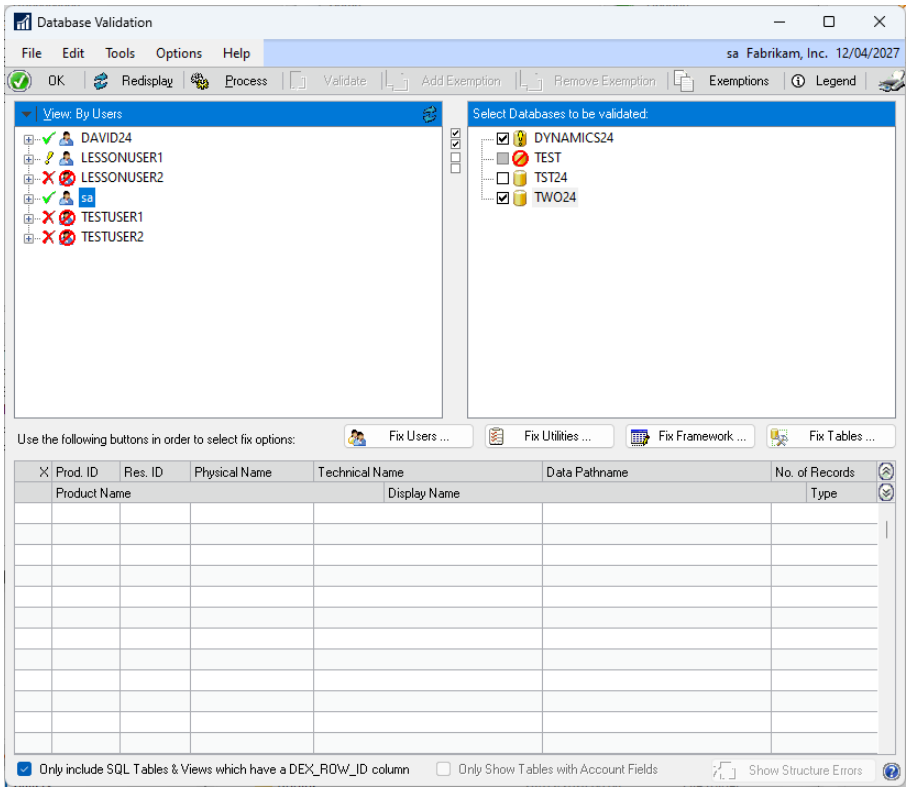
- Identify records for companies that are not installed
- Identify records for product dictionaries that are not installed

**Account Framework**:

- Identify Account Framework from Application Dictionary
- Identify Account Framework from setup tables in System Database
- Identify Account Framework from GL_Account_MSTR (GL00100) table in each Company Database

The results of the checks are then displayed when the window opens.

*Please note that this screenshot is intentionally showing errors. A system with no errors will have green ticks next to all of the users, companies and databases shown in the top left and top right panes. See the final screenshot at the end of the Using Database Validation section below.*




The following is a description of the individual fields on the window:

### *OK Button*

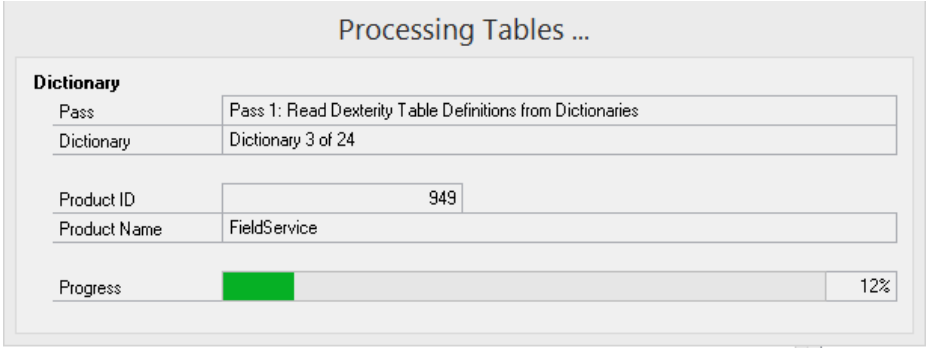
This button closes the window, saving the list of Exempted tables.

### *Redisplay Button*

This button restarts Datsabase Validation window and re-runs the initial checks listed above.

### *Process Button*

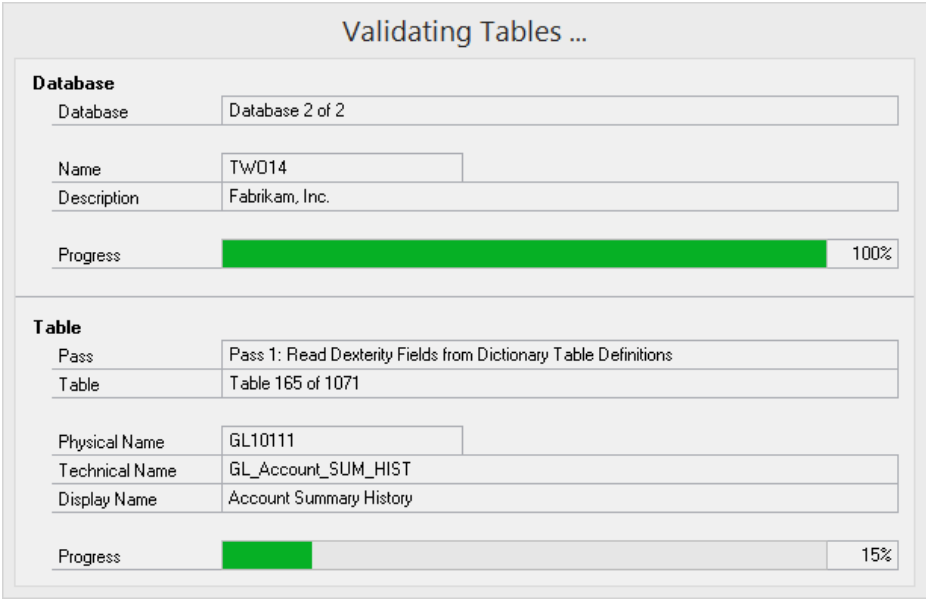
When this button is pressed, Database Validation reads all the table definitions from the installed Dexterity product dictionaries. It then reads the tables and views from the selected SQL Server databases and identifies which tables and views exist in both SQL Server and the product dictionaries and those only found in one location.




*To avoid Database Validation incorrectly classifying tables or views as missing, please ensure that the workstation used to run Database Validation has all installed product dictionaries installed.*

### *Validate Button*

When this button is pressed, Database Validation will compare the data structures for tables in the selected database tables against the matching table definitions in the product dictionaries and identify any differences.




*Running the Validation process can take a while to run as it has to compare the data structures for all the selected tables in multiple databases. This process runs in the foreground and you will not be able to perform other tasks while it is running.*

### *Add Exemption Button*

This button is enabled when one or more missing tables or views are selected. It can be used to quickly add tables and/or views to the Exemptions list.

*Remove Exemption Button*

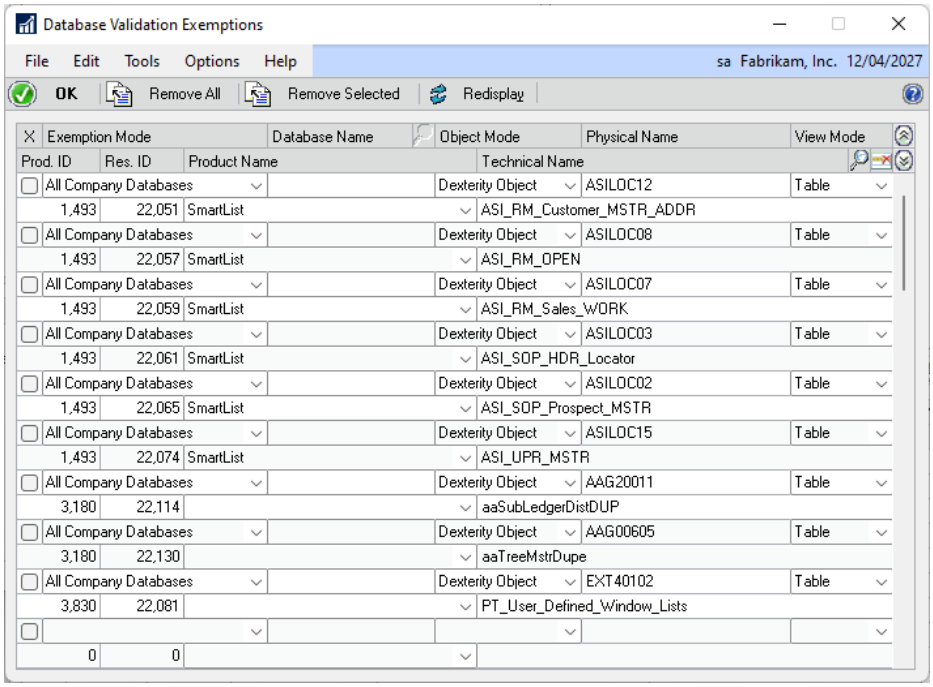
This button is enabled when one or more exempt tables or views are selected. It can be used to quickly remove tables and/or views from the Exemptions list.

*Exemptions Button*

This button opens the Database Validation Exemptions window so that manual changes can be made to the Exemptions list.

The Exemptions list is populated with some known tables and views automatically, but you can manually add additional tables and views which will then be excluded from the validation process.

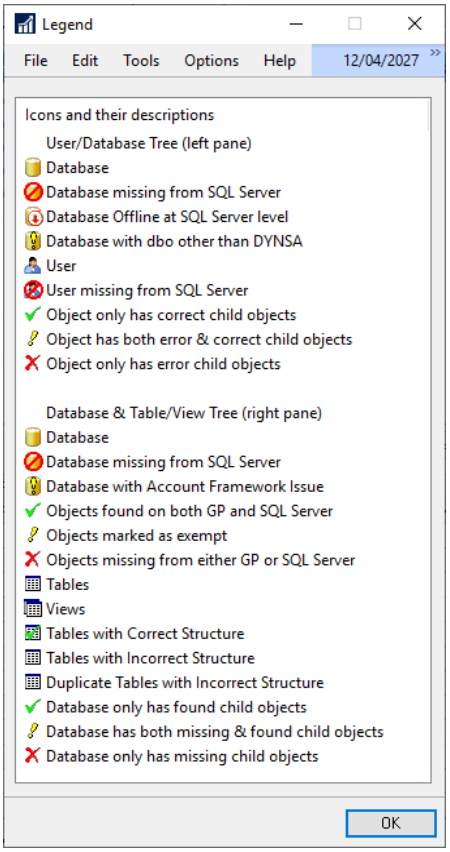
Exemption Mode can be from the System Database, All Company Databases or for Specified Company Database. Object Mode can be Dexterity Objects or SQL Objects and the View Mode can be Table or View. When specifying a Dexterity Object, you can select the object by entering its physical name or by selecting the Product Name and entering the Technical Name.




To remove entries from the Exemptions list, you can remove individual lines, or select a number of lines and Remove Selected or Remove All.

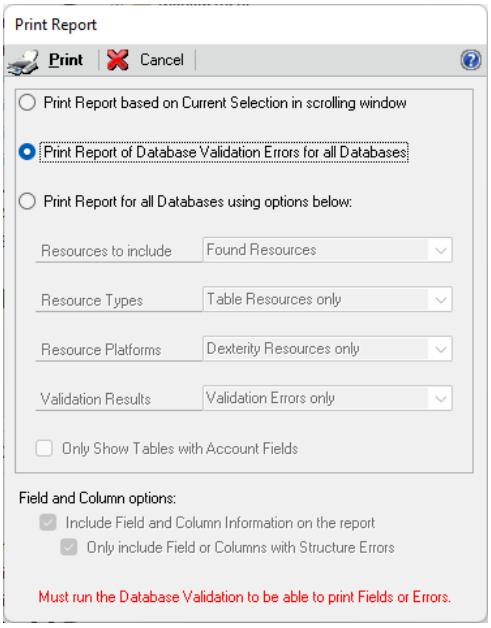
### *Legend Button*

This button opens the Database Validation Legend window which explains the various icons and indicators used by Database Validation.



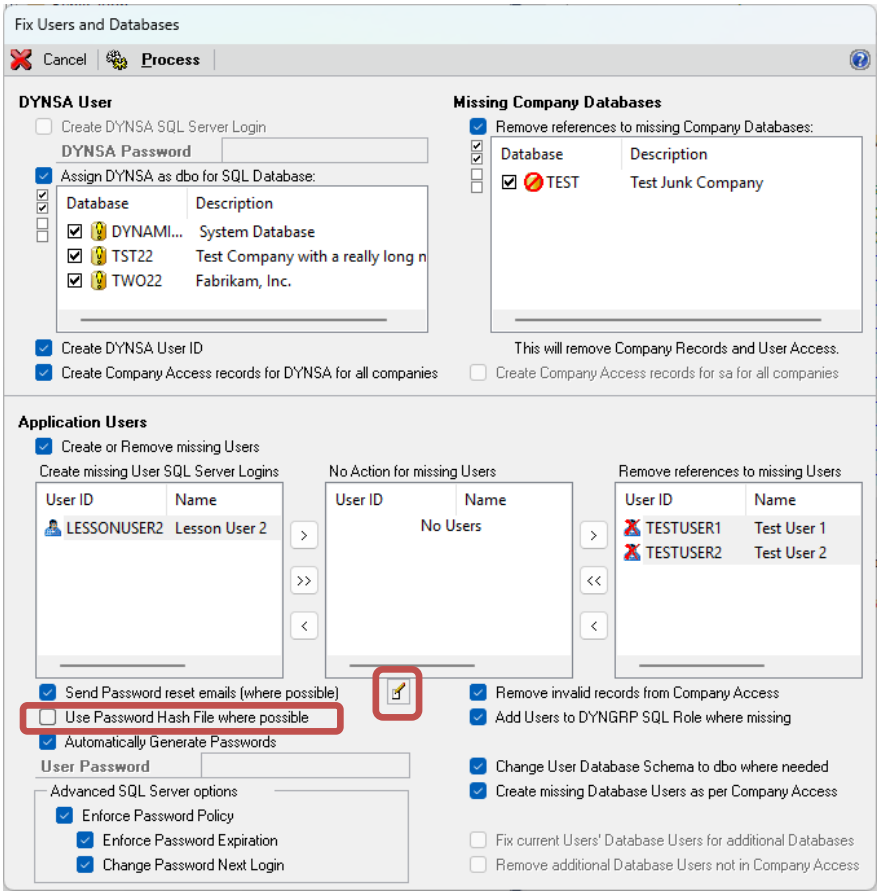

### *Print Button*

This button opens the Print Report window where you can select what information to include on the printed report.



### *Fix Users Button*

This button opens the Fix Users and Databases window. The various sections of this window will only be enabled if there are errors for that section to be resolved. To the left of the top two panes on the window are Mark All and Mark None buttons which can be used to quickly change the checkbox selections in that pane. If more than one item is highlighted, Mark All and Mark None will be applied to the highlighted items.




The Edit button can be used to open the Password Reset Email Settings window to edit the email sent when resetting passwords.

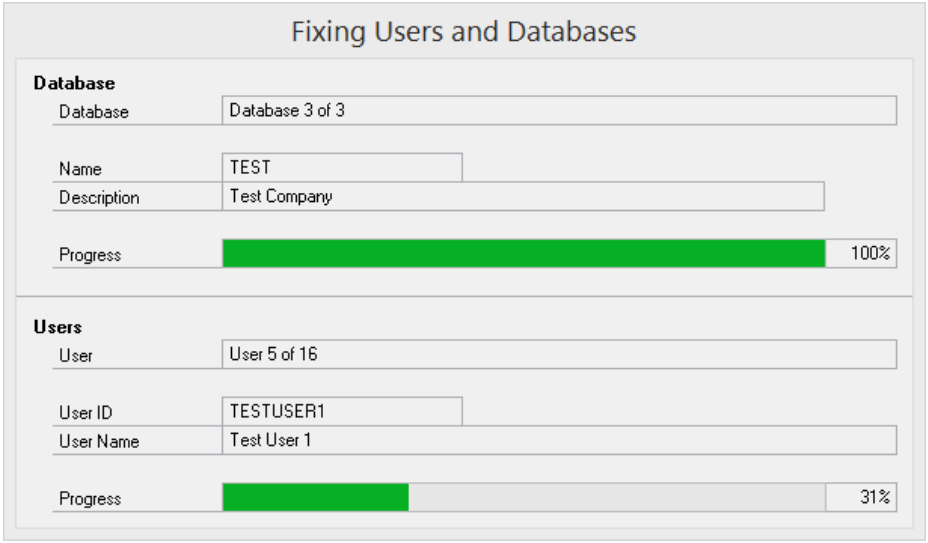
*Password Reset Emails can be sent automatically when Database Validation knows the User's email address. Use the User Setup Additional Information window to enter this and other user related data.*

Use the Use Password Hash File where possible option to use the same passwords for each user based on a previously exported Password Hash File.

*The Password Hash File method of restoring the same passwords for users will only work if the new SQL Server machine has been given the same name as the old SQL Server machine. If the SQL Server's machine name has changed, Microsoft Dynamics GP's password encryption algorithm means that the passwords will not work.*

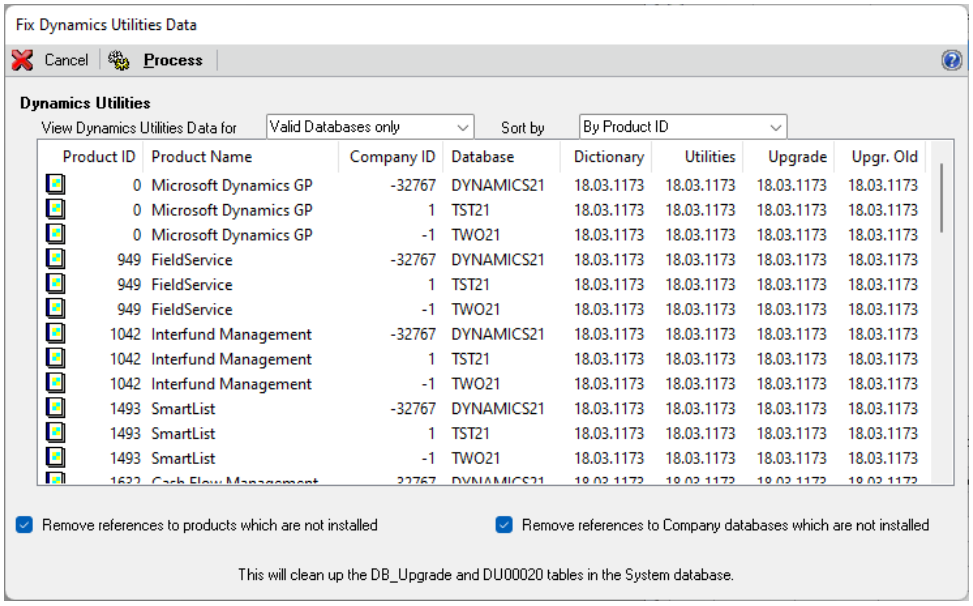
The Password Hash File can be exported with Database Validation on the old server using Options >> Capture Password Hash File menu.

The Process Button will be enabled if any fixing options have been selected. Click the Process Button to fix the selected issues.




### *Fix Utilities Button*

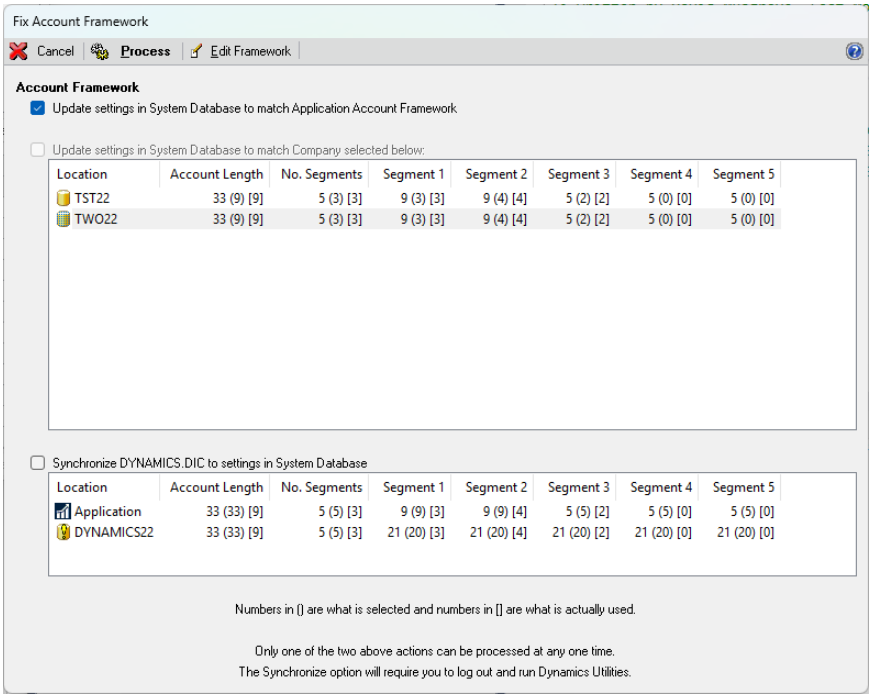
This button opens the Fix Utilities window.




The Process Button will be enabled if any fixing options have been selected. Click the Process Button to fix the issues found.

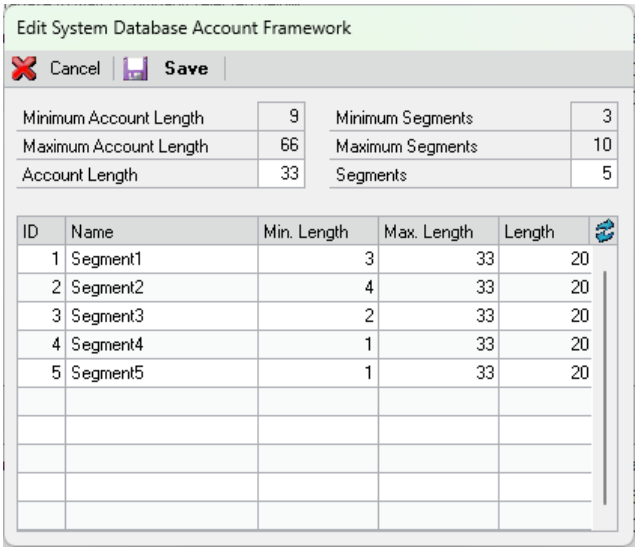
### *Fix Framework Button*

This button opens the Fix Account Framework window.




The Process Button will be enabled if any fixing options have been selected. Click the Process Button to fix the issues found.

Use the Edit Framework Button to edit the System Database Account Framework. This option can be used to change the Account Framework.




After editing the Account Framework, you will need to Synchronize DYNAMICS.DIC to match the settings in the System Database. Then Processing and Fixing Tables can used to adjust table structures.

### *Fix Tables Button*

This button opens the Fix Tables window. The various sections of this window will only be enabled if there are errors for that section to be resolved.

The table errors are divided into four sections:

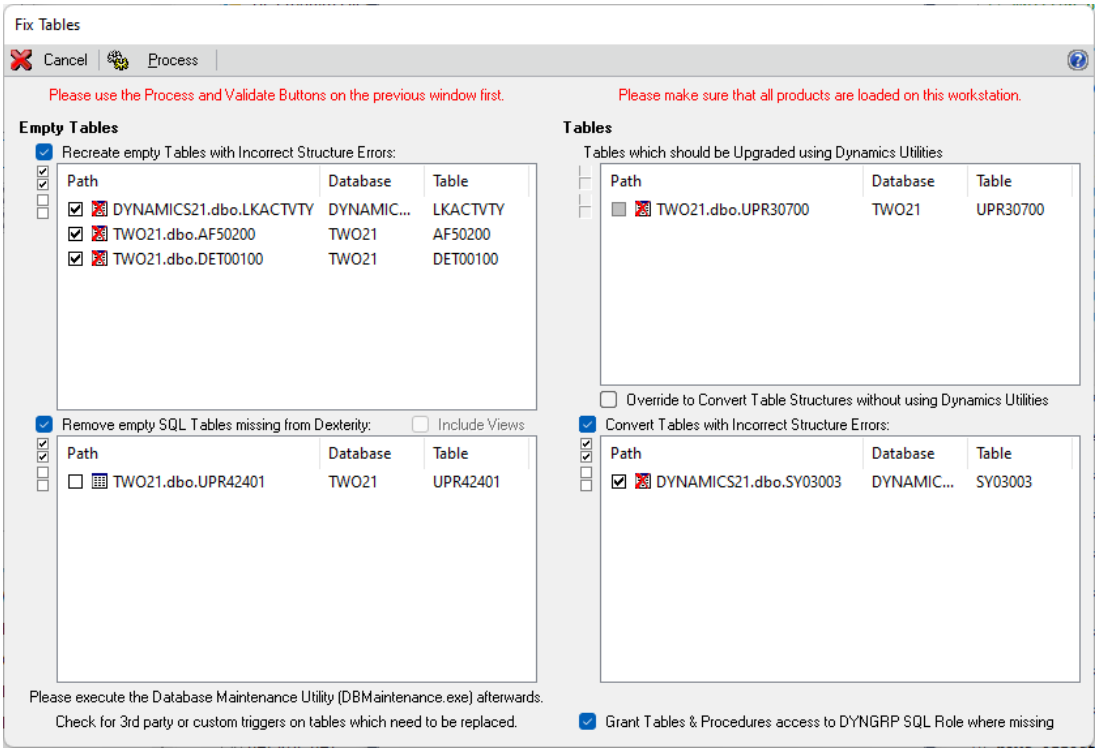
- **Empty tables with incorrect Structure Errors**. These tables can be dropped and recreated without needing to consider any existing data.

- **Empty SQL Tables missing from Dexterity dictionaries**. These tables can possibly be removed as they have no data and appear not to be used. They could be from a product that was installed and never used and has since been removed.

- **Tables containing data with incorrect Structure Errors which can be upgraded with Dynamics GP Utilities**. These are tables with date that have structure errors, however there is a conversion available via Dynamics GP Utilities. You should attempt to upgrade with Dynamics GP Utilities first as this should perform the proper conversion steps and might update data as part of the upgrade.

*The Override to Convert Table Structures without using Dynamics Utilities option should be used with caution as it will bypass any additional conversion steps that might have been performed by Dynamics GP Utilities.*

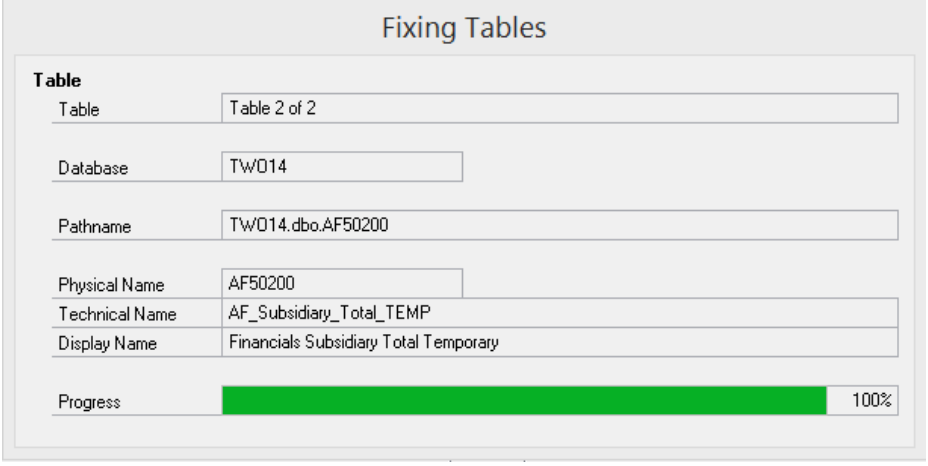
- **Tables containing data with incorrect Structure Errors which cannot be upgraded with Dynamics GP Utilities**. These are tables with data that have structure errors, but there is no Dynamics GP Utilities conversion available. You can use Database Validation to automatically backup the data, drop and recreate the tables and restore the data.




To the left of each pane on the window is a Mark All and Mark None button which can be used to quickly change the checkbox selections of the tables in that pane. If more than one item is highlighted, Mark All and Mark None will be applied to the highlighted items.

*Ensure you have a backup of all Microsoft Dynamics GP databases before running the Fix Tables process. If you are unsure about running any Fix Tables process, please contact your support consultants to discuss.*

The Process Button will be enabled if any fixing options have been selected. Click the Process Button to fix the selected issues.




*After running the Fix Tables process, it is recommended to run the Database Maintenance Utility (DBMaintenance.exe) to update or create any additional SQL Server resources.*

### *Only include SQL Table & Views which have a DEX_ROW_ID column*

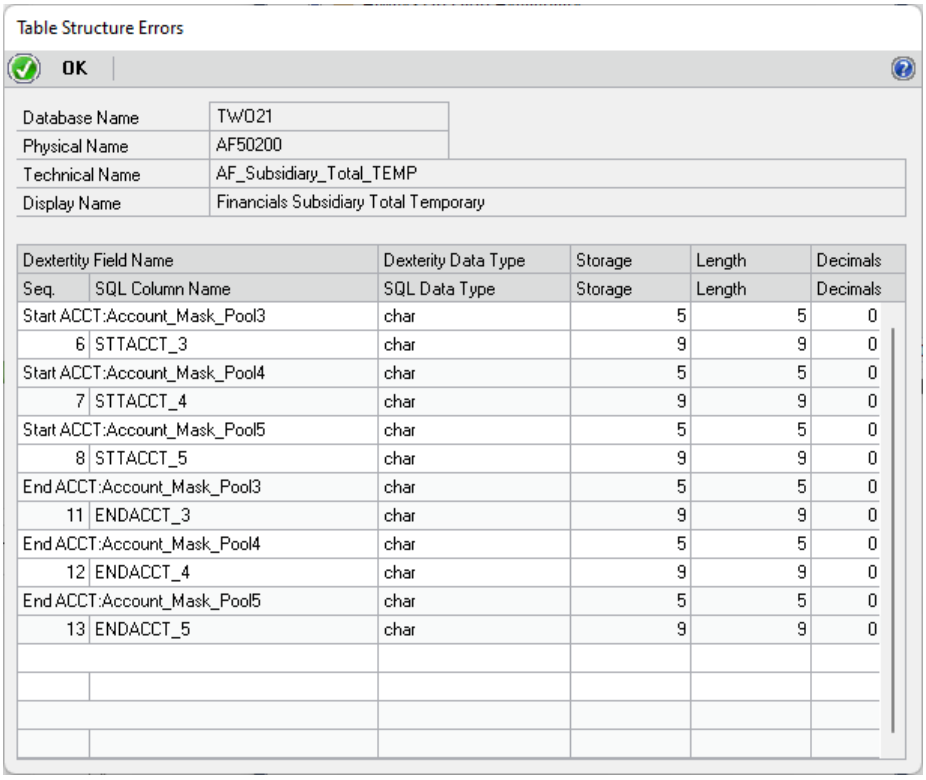
This option limits the SQL Tables and Views reviewed to only include ones that include a DEX_ROW_ID column. Keeping this option selected prevents Database Validation looking at additional SQL objects that are not used with Dexterity product dictionaries.

### *Only Show Tables with Account Fields*

This option filters the scrolling window to only include tables which include an Account Number field. This is useful when looking for tables with Account Framework issues.

### *Show Structure Errors Button*

This button opens the Table Structure Errors window. This window details the database structure errors for the selected table.

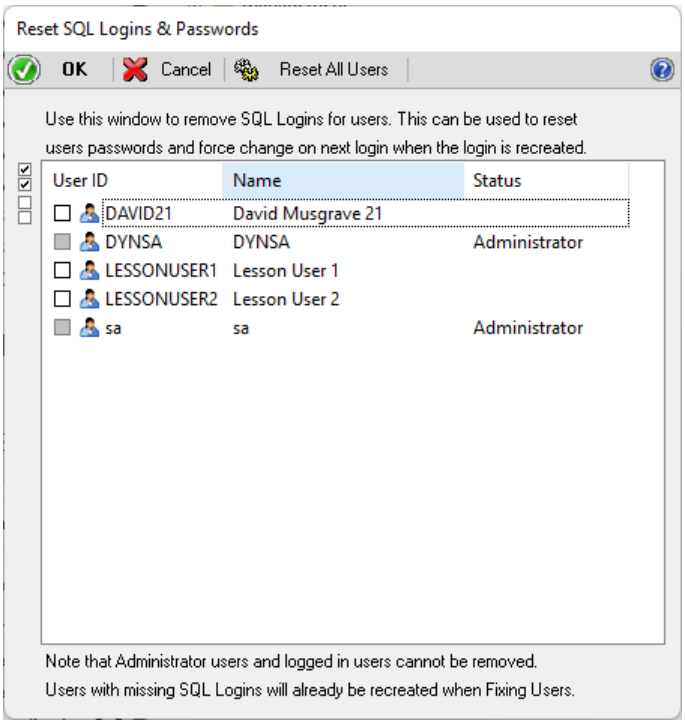



| Database Name | TWO21 |
|---|---|
| Physical Name | AF50200 |
| Technical Name | AF_Subsidiary_Total_TEMP |
| Display Name | Financials Subsidiary Total Temporary |

| Dextertity Field Name | | Dexterity Data Type | Storage | Length | Decimals |
|---|---|---|---|---|---|
| Seq. | SQL Column Name | SQL Data Type | Storage | Length | Decimals |
| Start ACCT:Account_Mask_Pool3 | | char | 5 | 5 | 0 |
| 6 | STTACCT_3 | char | 9 | 9 | 0 |
| Start ACCT:Account_Mask_Pool4 | | char | 5 | 5 | 0 |
| 7 | STTACCT_4 | char | 9 | 9 | 0 |
| Start ACCT:Account_Mask_Pool5 | | char | 5 | 5 | 0 |
| 8 | STTACCT_5 | char | 9 | 9 | 0 |
| End ACCT:Account_Mask_Pool3 | | char | 5 | 5 | 0 |
| 11 | ENDACCT_3 | char | 9 | 9 | 0 |
| End ACCT:Account_Mask_Pool4 | | char | 5 | 5 | 0 |
| 12 | ENDACCT_4 | char | 9 | 9 | 0 |
| End ACCT:Account_Mask_Pool5 | | char | 5 | 5 | 0 |
| 13 | ENDACCT_5 | char | 9 | 9 | 0 |

Click OK to close the window.

*The Database Validation window has an Options Menu which can be used to Reset User SQL Logins and Passwords. This option can be used for force a reset of selected users' or all users' passwords by removing their SQL Logins and allowing Database Validation to recreate them. It will not remove the SQL Login for any users currently logged into the system.*




If you wish to reset user passwords, view or change password policy settings or change a user's status, you might find the SQL Login Maintenance window a better approach as it does not remove the SQL Login and require it to be recreated. See the next section for more information.

The Mark All and Mark None buttons can be used to quickly change the checkbox selections of users. If more than one item is highlighted, Mark All and Mark None will be applied to the highlighted items.

## Using Database Validation

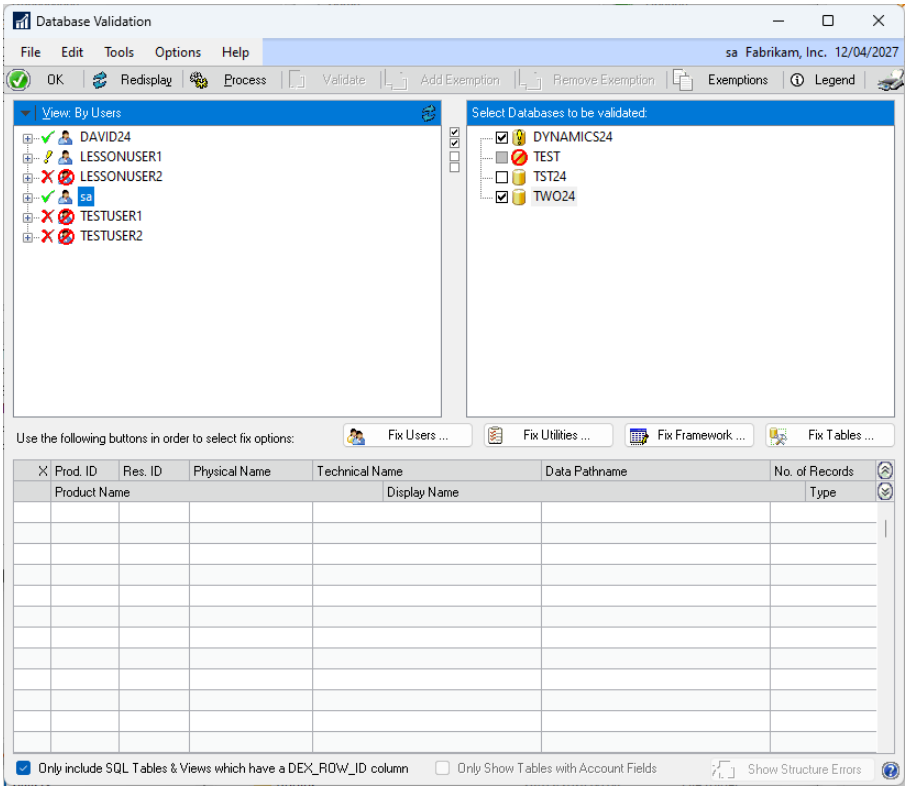
The following section explains the process of using Database Validation on your system.

1. Backup all system and company SQL Server databases.

2. Ensure that no other users are logged in.

3. Ensure that all Dexterity product dictionaries are installed on the current workstation.

4. After Database Validation has performed its initial checks, the window will open and display what issues it has found.

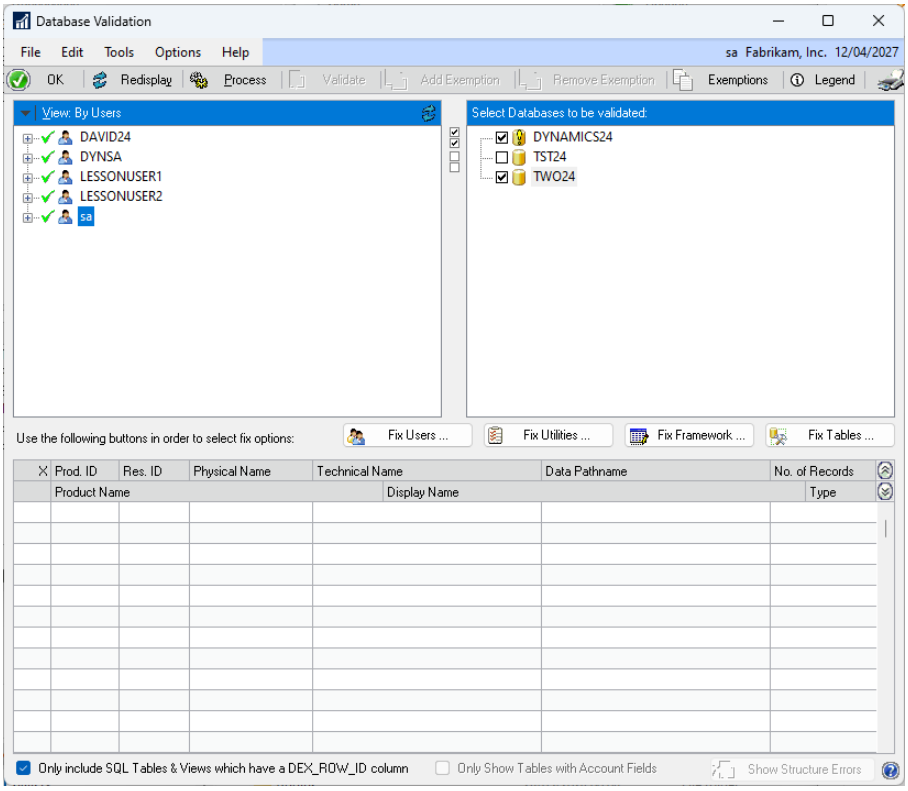



5. Use the Fix Users and Databases window to resolve any issues with Users and Databases. The window will refresh after the process.

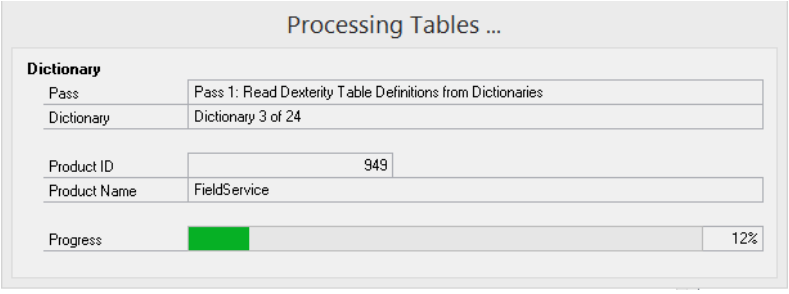
6. Use the Fix Utilities window to resolve any issues with the data in the Dynamics GP Utilities version tables. The window will refresh after the process.

7. Use the Fix Account Framework window to resolve any issues with the data Account Framework in the system. The window will refresh after the process.

8. Once all these issues are fixed, the Database Validation window show now display with no errors.



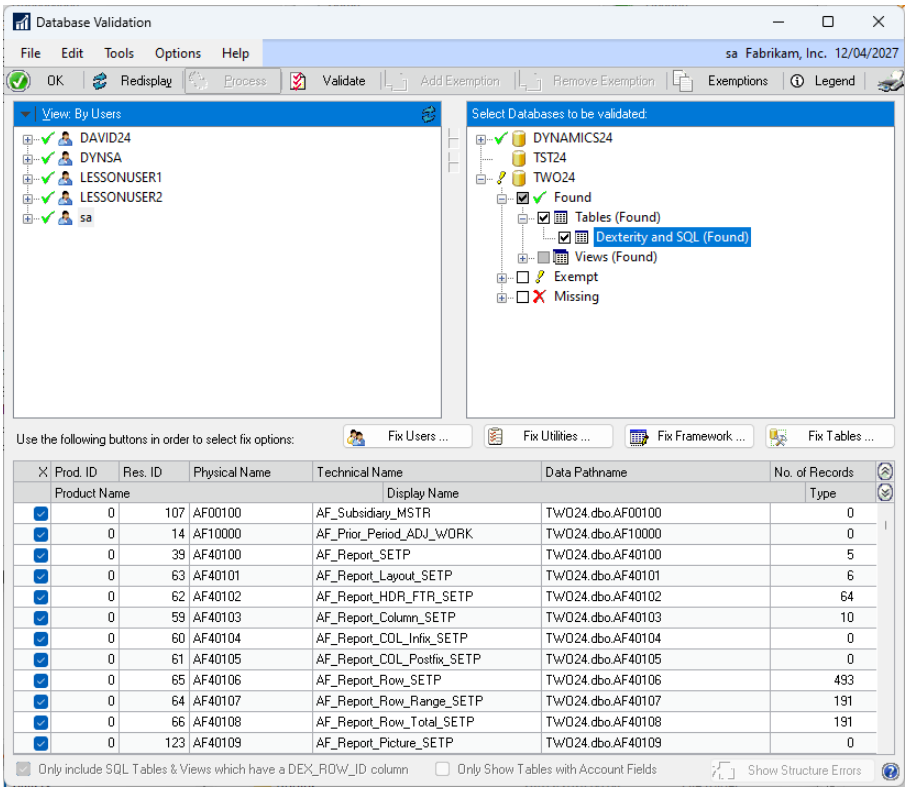

9. Click Process to read the tables and views from all Dexterity product dictionaries installed and match them to the tables and views in the selected SQL Server system database and company databases in the top right-hand pane.



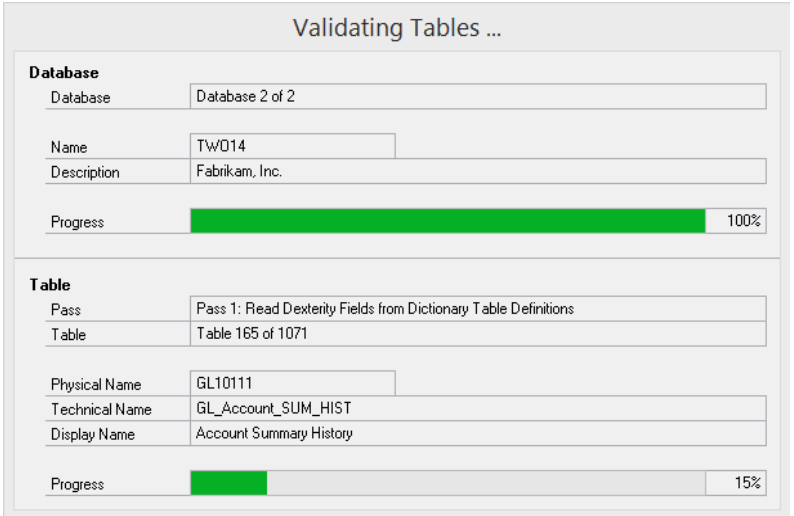

10. Once the processing has completed, you can explore the tree in the top right pane and see what tables and views have been found to exist in both the Dexterity product dictionaries and the SQL Server databases. The scrolling window in the bottom half of the window will display the tables depending on the node selected in the top right pane.

11. You can select Missing tables and views and add them to the Exemptions using the Add Exemptions button, so they don't show as missing next time.

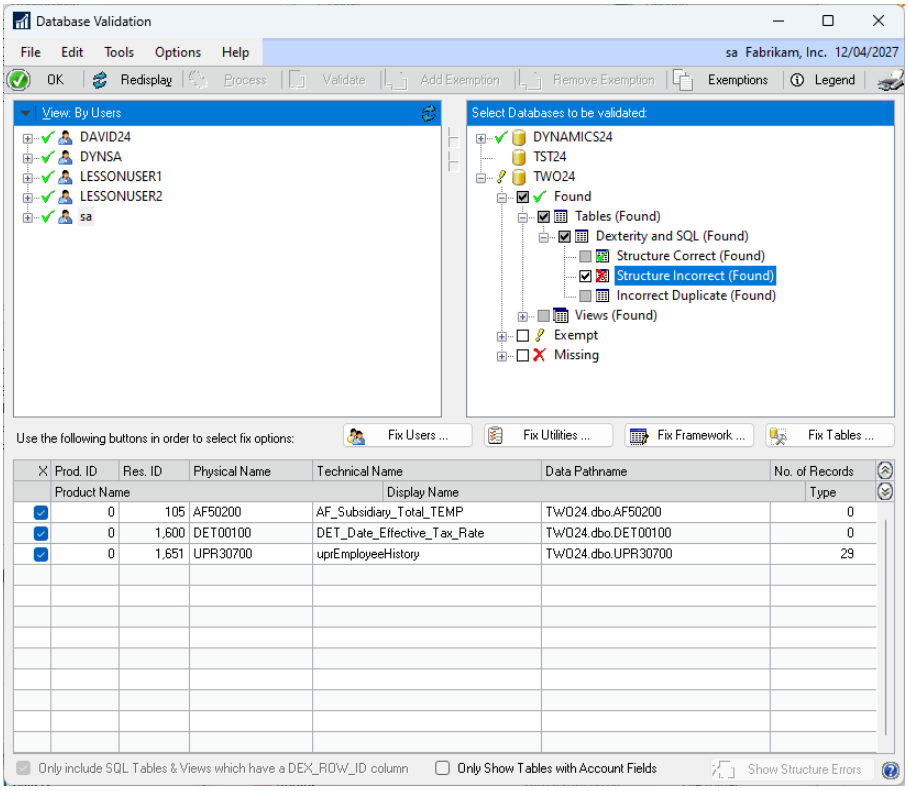
12. You can select all or some of the Found Tables in the databases. The selection can be made my clicking on the node checkboxes or by selecting a node and then clicking on the tables in the scrolling window. Clicking on the node checkboxes in the tree can be used to mark all or mark none.



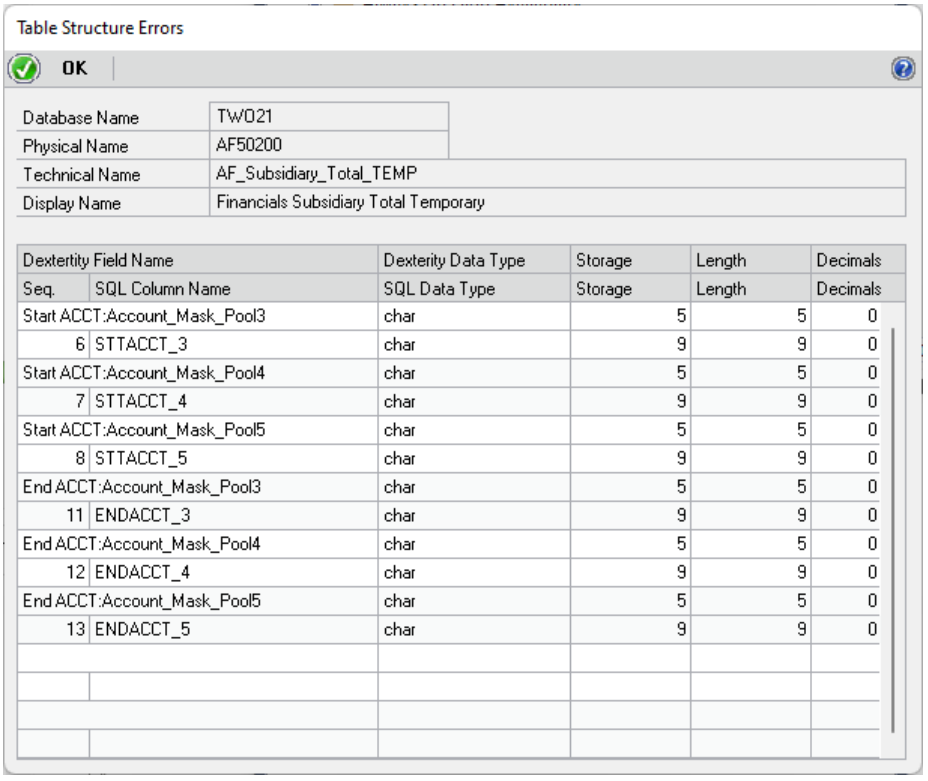

13. Click Validate to compare the table structures for the selected tables found in both Dexterity product dictionaries and SQL Server databases. This process can take some time depending on how many tables are being checked. It runs multiple passes to complete the process.



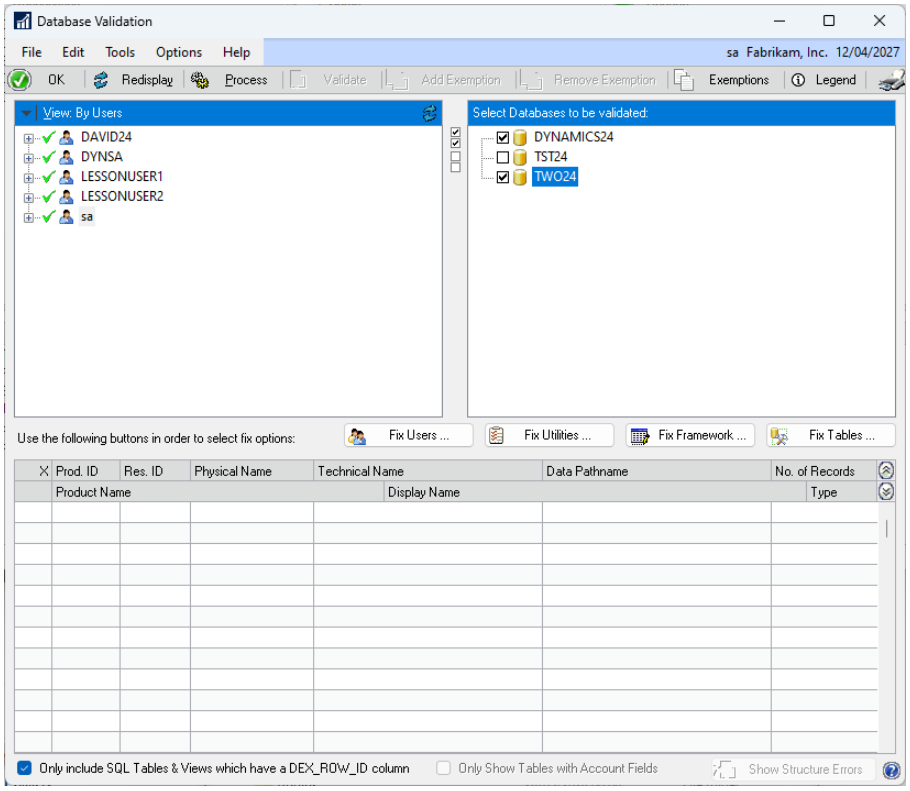
14. After the Validation process is complete, an optional report can be printed to show the results. You can use the tree to explore the results and show the tables which have been identified as Structure Incorrect.




15. If you wish to see the detail of the Structure Errors, select the individual table in the scrolling window and click the Show Structure Errors button.




16. Use the Fix Tables window to resolve any issues with Tables. The window will refresh after the process.



17. When closing the Database Validation window, print or save the Database Validation Log report generated with all the actions processed by Database Validation.

18. Exit Dynamics GP and run the Database Maintenance utility against all system and company databases.

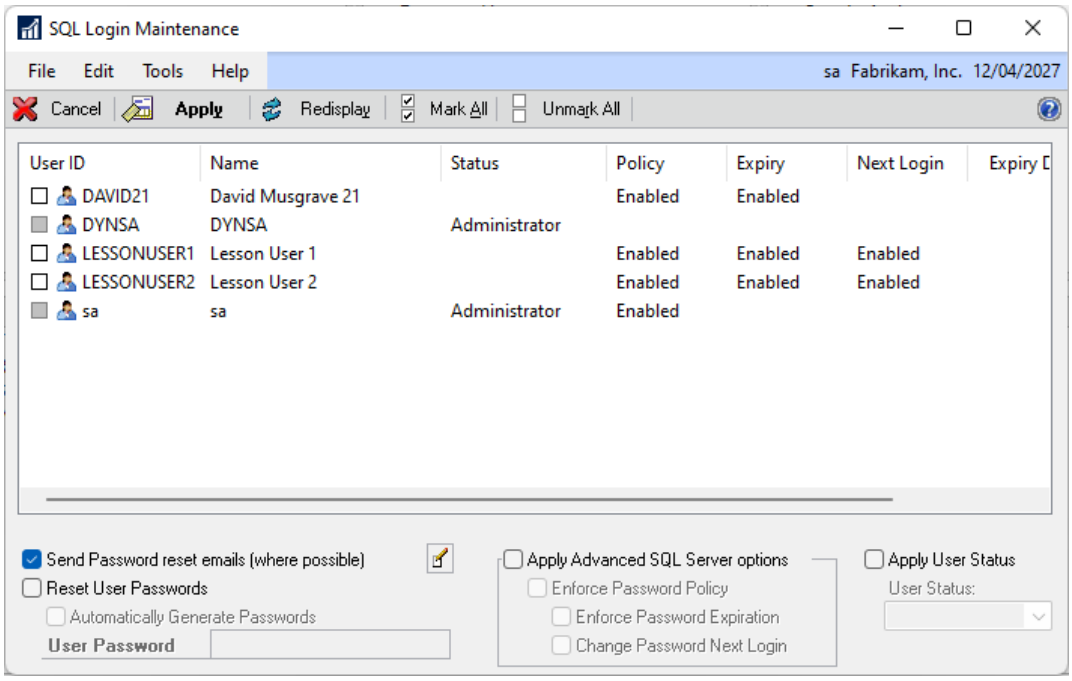
19. Make a second backup of all system and company SQL Server databases of your validated system.

## SQL Login Maintenance

You can open the SQL Login Maintenance window by selecting SQL Login Maintenance from the Utilities section of the GP Power Tools Area Page or by selecting Database >> SQL Login Maintenance from the Options button drop list on the main window. This is an Advanced Mode feature.

The SQL Login Maintenance window is designed to provide a simple method to reset user passwords, view or change password policy settings or make users inactive or active for multiple users at one time.

*The Database Validation has an option to Reset User SQL Logins and Passwords. Using the SQL Login Maintenance window instead resets the password without removing and recreating SQL Logins. If there is a problem with the SQL Login you can use Database Validation to fix it before using SQL Login Maintenance.*




The following is a description of the individual fields on the window:

### *User List*

This list shows all the users in the system with their current status and password policy settings. Double clicking on a user will open User Setup for the selected user. From the User Setup window, you can check the User Setup Additional Information window and ensure the User Email Address is populated. Use the checkboxes to select the users you wish to apply any changes to.

### *Send Password changed emails*

When this checkbox is selected, SQL Login Maintenance will send emails to users when resetting their password as long as the User Email Address is populated for the user in the User Setup Additional Information window.

The Edit button can be used to open the Password Reset Email Settings window to edit the email sent when resetting passwords.

*Password Reset Emails can be sent automatically when SQL Login Maintenance knows the User's email address. Use the User Setup Additional Information window to enter this and other user related data.*

*Reset User Passwords*

Select this checkbox if you want to reset the passwords for the selected users.

*Automatically Generate Passwords*

Select this checkbox if you want GP Power Tools to generate individual passwords for each user.

*User Password*

If not generating passwords, use this field to enter a single password to user for all users.

*Apply Advanced SQL Server options*

Select this checkbox if you want to change Advanced SQL Server options for the selected users.

*Enforce Password Policy*

Select this checkbox to update users to enable the system's password policies.

*Enforce Password Expiration*

Select this checkbox to update users to enable the password expiration policy.

*Change Password Next Login*

Select this checkbox to force users to change their password on next login. This option can only be used when resetting passwords.

*Apply User Status*

Select this checkbox if you want to change User Status settings for the selected users.

*User Status*

Use this drop down list to select if users should be marked Active, Inactive or as Lesson Users.

The following is a description of the additional buttons on the window:

*Cancel Button*

This button closes the window without taking any further actions.

*Apply Button*

This button will apply the selected setting changes to the selected users.

*Redisplay Button*

This button will refresh the window with the current users and their settings.

*Mark All Button*

Use this button to mark all the users (or all highlighted users) which can be updated.

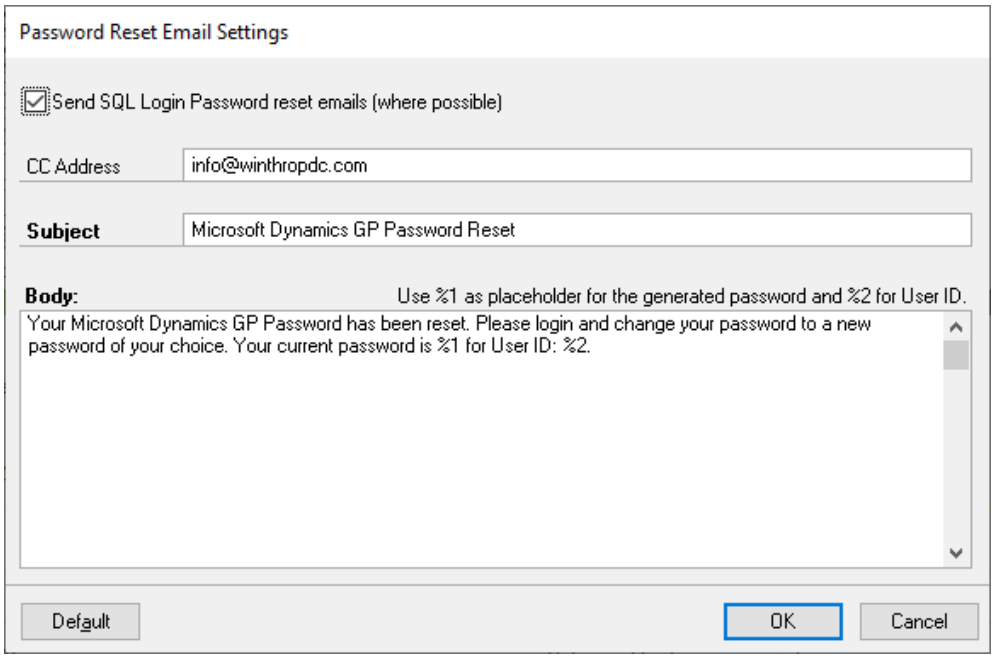
*Unmark All Button*

Use this button to unmark all the users (or all highlighted users) which can be updated.

## Password Reset Email Settings

You can open the Password Reset Email Settings window by selecting Password Reset Email Settings from the Utilities section of the GP Power Tools Area Page or by selecting Database >> Password Reset Email Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

The Password Reset Email Settings window controls the settings for sending emails when passwords are reset from the Database Validation, SQL Login Maintenance or User Setup windows.




The following is a description of the individual fields on the window:

*Send SQL Login Password reset emails*

This checkbox enables sending of emails when passwords are reset if a user's email address or CC email address has been entered.

*Password Reset Emails can be sent automatically when the User's email address is known. Use the User Setup Additional Information window to enter this and other user related data.*

*CC Address*

This field contains the email address used when sending emails. If the CC address has been entered, but there is no user email address available, the email will just be sent to the CC address.

*Subject*

This field contains the subject line to be used when sending emails.

*Body*

This field contains the body text to be used when sending emails. Use the placeholder %1 for the password and %2 for the User ID.

*Default Button*

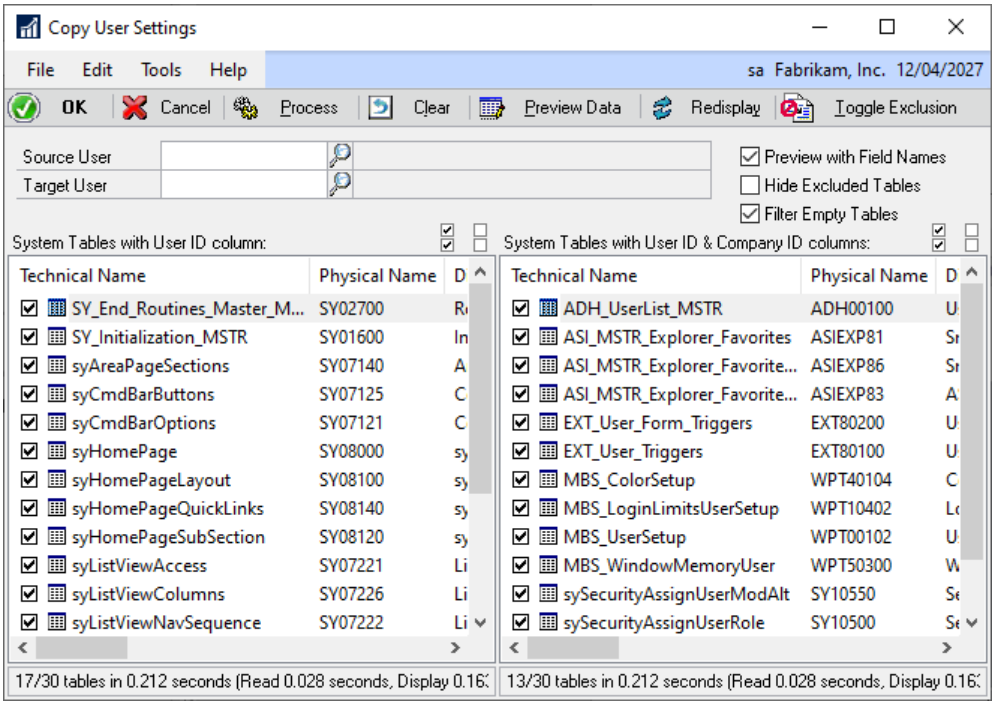
Use this button to restore the default Subject and Body settings.

## Copy User Settings

You can open the Copy User Settings window by selecting Copy User Settings from the Utilities section of the GP Power Tools Area Page or by selecting Database >> Copy User Settings from the Options button drop list on the main window. This is an Advanced Mode feature.

The Copy User Settings window allows copying of user settings in the system database from a source User ID to a target User ID. All tables containing the User ID field are listed in the left-hand pane and all tables containing both the User ID and Company ID fields are listed in the right-hand pane.

Use this window after creating a new user to transfer all system settings (including any 3rd party products) from an existing user to the newly created user.




The following is a description of the individual fields on the window:

*Source User ID*
Select the source User ID to copy the user records from.

*Target User ID*
Select the target User ID to receive the copied records.

*Preview with Field Names*
This checkbox controls if the Dexterity Technical Names or SQL Physical Names are used as the column headers when previewing data.

*Hide Excluded Tables*

This checkbox controls whether the excluded tables are shown in the table lists. When the excluded tables are not hidden, the Toggle Exclusion Button is shown to allow the excluded tables to changed. This field defaults to selected to hide the excluded tables and the Toggle Exclusion Button.

*Filter Empty Tables*

This option will hide empty tables from the table lists as there is no data to copy. When a Source User ID has been entered, this option is turned on to hide any tables with no data for the Source User ID.

*System Tables with User ID column*

This table list shows the tables from the system database which contain the User ID field.

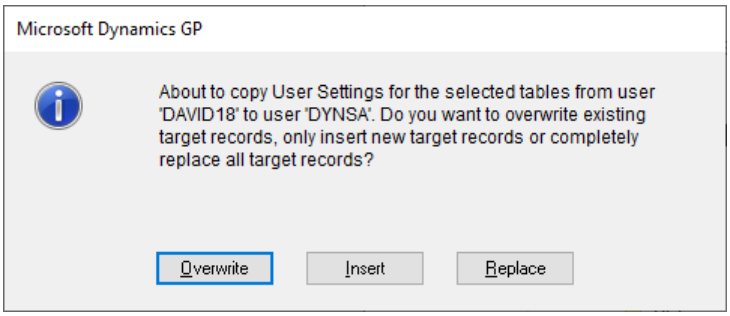
*System Tables with User ID & Company ID column*

This table list shows the tables from the system database which contain the User ID and Company ID fields.

The following is a description of the additional buttons on the window:

*Process Button*

Click this button to start the copy user settings process. If source and target users are entered the following dialog will be displayed to control how the copy process will proceed.




- Overwrite: Copy all settings from source user to target user overwriting any existing records. Records in the target user but not in the source user will remain.
- Insert: Insert missing settings from source user to target user. Existing records on the target user will be unchanged.
- Replace: Copy all settings from source user to target user replacing any existing records. Records in the target user but not in the source user will be removed.

*Clear Button*

Use this button to reset the window back to default settings.

### *Preview Data Button*

This button will open the SQL Execute Setup window to preview the data for the selected fields in the SQL table. The Preview with Field Names option controls if the Dexterity Technical Names or SQL Physical Names are used as the column headers.

*Previewing data uses the SQL Execute Setup window to display the data and so needs the Developer Tools module registered.*

### *Redisplay Button*

Use this button refresh the table lists.

### *Toggle Exclusion Button*

Use this button to turn on and off exclusion of the selected table. This button and the excluded tables will be hidden if the Hide Excluded Tables checkbox is selected.

### *Mark All Buttons*

Use this button to mark all of the tables (or all highlighted tables) in the table list.

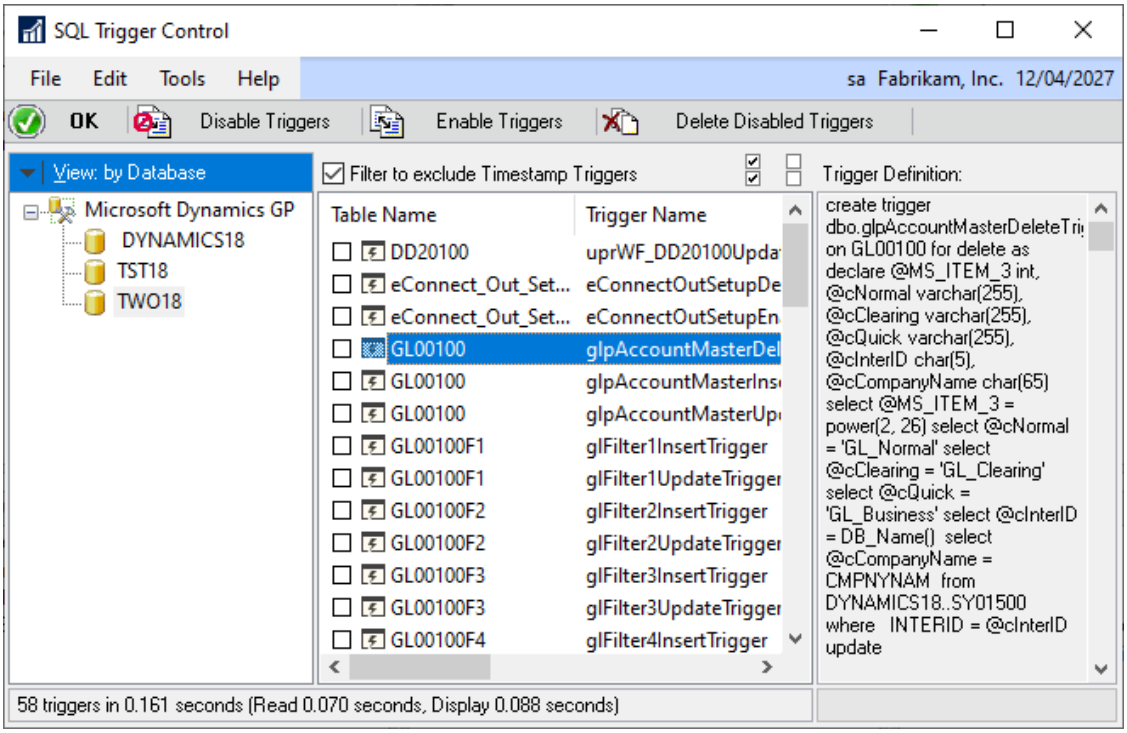
### *Unmark All Buttons*

Use this button to unmark all of the tables (or all highlighted tables) in the table list.

## SQL Trigger Control

You can open the SQL Trigger Control window by selecting SQL Trigger Control from the Utilities section of the GP Power Tools Area Page or by selecting Database >> SQL Trigger Control from the Options button drop list on the main window. This is an Advanced Mode feature.

The SQL Trigger Control window is used to disable, enable and delete SQL table triggers for troubleshooting or system maintenance purposes. It can be used cleanup after a customization using SQL triggers is removed and does not uninstall correctly.




The following is a description of the individual fields on the window:

*Database Tree*

The left-hand pane displays the databases in the system either by database name or by company name.

*Trigger List*

The middle pane displays the SQL table triggers for the selected database. Scroll to the right to see additional information about the triggers. This list can be filtered to remove the Dexterity Timestamp triggers using the Filter to exclude Timestamp Triggers checkbox.

*You can multi-select triggers in the Trigger List using the control and shift keys. The Mark All and Unmark All buttons will mark and unmark just the selected triggers when more than one trigger is selected.*

*Trigger Definition*

When a single trigger is selected, the definition for the trigger is shown in the right-hand pane.

*Filter to exclude Timestamp Triggers*
Select this checkbox to hide the Dexterity Timestamp triggers from the Trigger List.

The following is a description of the additional buttons on the window:

*Disable Triggers Button*
Use this button to disable the marked triggers.

*Enable Triggers Button*
Use this button to enable the marked triggers.

*Delete Disabled Triggers Button*
Use this button to delete the marked disabled triggers. Triggers must be disabled before they can be deleted.

*Mark All Button*
Use this to mark all the triggers or if more than one trigger is selected, it will mark the selected triggers.

*Unmark All Button*
Use this to unmark all the triggers or if more than one trigger is selected, it will unmark the selected triggers.

## Note Fix Utility

You can open the Note Fix Utility window by selecting Note Fix Utility from the Utilities section of the GP Power Tools Area Page or by selecting Database >> Note Fix Utility from the Options button drop list on the main window. This is an Advanced Mode feature.

The Note Fix Utility window is used to identify and fix issues with Record Note and the Note Index values used to link setup, cards and transaction records to the Record Note Master (SY03900) table. Issues can include:

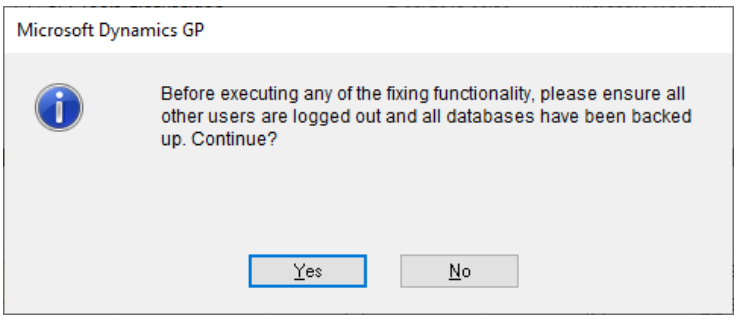
- The Company's Next Note Index stored in the Company Master (SY01500) table in the system database is lower than the maximum used Note Index in the company. Often caused by restoring Databases separately.
- Zero Value Note Indexes where records have no Note Index value assigned to them. Often caused by importing data and not assigning the Next Note Index.
- Duplicate Note Indexes where the same Note Index has been given to more than one record in the system. Often caused by the Company's Next Note Index being incorrect due to restoring Databases separately.
- Used Duplicate Note Indexes where the same Note Index has been given to more than one record in the system AND that note has been used by one (or more) of those records. The result is cross linked notes where the note shows up for multiple records in the system.
- Note with no Note Index when there is a record in the Record Note Master (SY03900) table with a zero value for the Note Index column.
- Orphaned Notes where records in the Record Note Master (SY03900) table cannot be matched with any other records in the system. This could happen if an ISV product is using a different field to store the Note Index or if data has been removed or archives without removing the used Notes.

*If Orphaned Notes are showing due to an ISV product using a different field to store the Note Index values, this issue can be resolved by adding the appropriate field to the Field list using the Edit Fields Button below.*

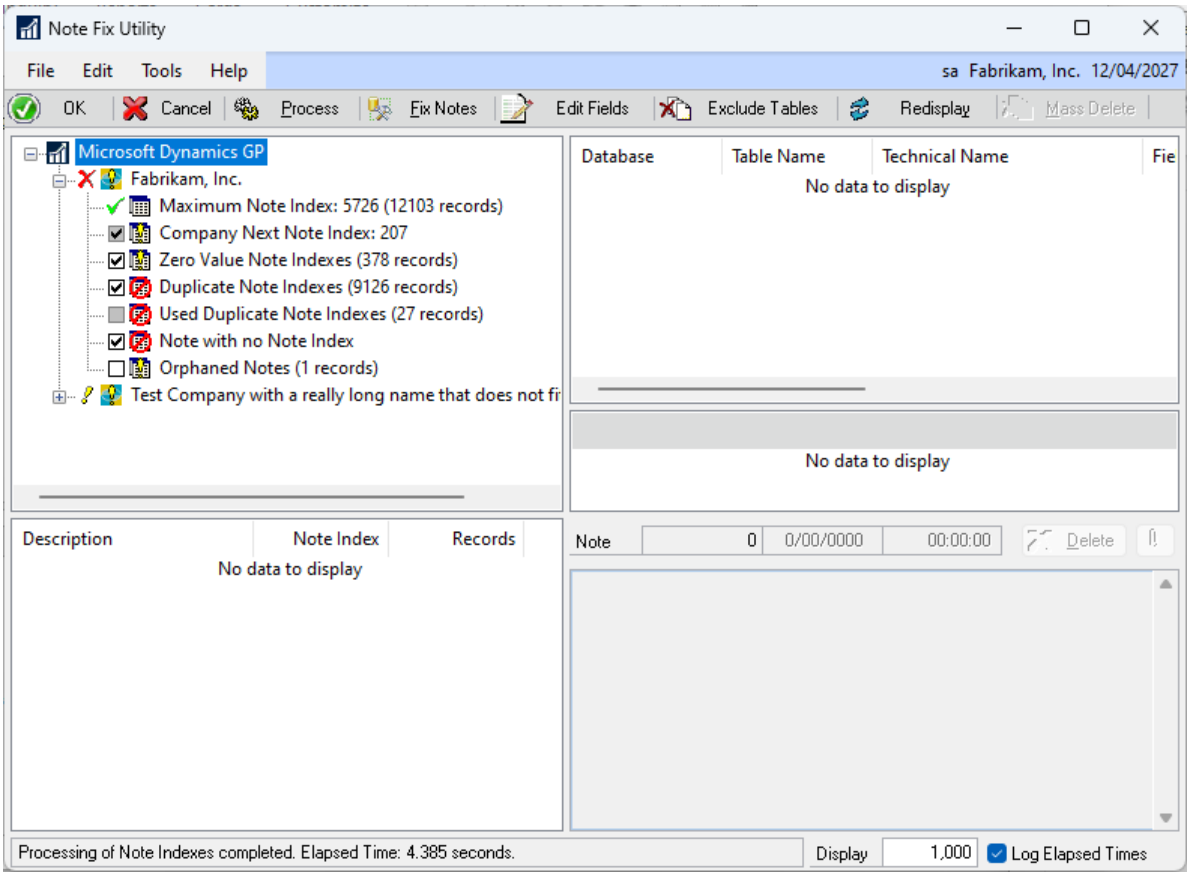
Running the Note Fix Utility after restoring databases can resolve any potential issues before they occur.

Before the window opens you will be reminded to ensure that all users are logged out and that your company and system databases have been backed up before executing any of the fixing functionality.

*Running the Note Fix Utility checks to identify issues is read-only and does not require backups or exclusive use of the system.*




When the window opens it will show the companies in the system with the current company selected for processing. Once the processing has run, you will be able to see if there are any issues that need to be resolved.




The following is a description of the individual fields on the window:

*Company Tree*

The top left-hand pane displays the companies in the system.

*Note Index List*

The bottom left-hand pane displays the Note Index values for the selected issue node in the Company Tree.

*Table List*

The top right-hand pane displays the tables for the selected Note Index value in the Node Index List.

*Record List*

The middle right-hand pane displays SQL Table Record for the selected Table Record in the Table List, for Used Duplicate Note Indexes mode only.

*Note Fields*

The bottom right-hand pane displays the selected Note Index, along with the last modified Date and Time and the Note text field contents.

*Delete Button*

The delete button can be used to manually delete the displayed note record, for Used Duplicate Note Indexes and Orphaned Notes modes only.

*Display Records*

This value is used to limit the amount of data displayed in the Note Index List and Table List. If background processing is available, it will be used when populating the lists and will allow up to the value of this field times 100 records to be displayed. If background processing is not available, foreground processing will be used to populate the lists and will allow up to the value of this field records to be displayed.

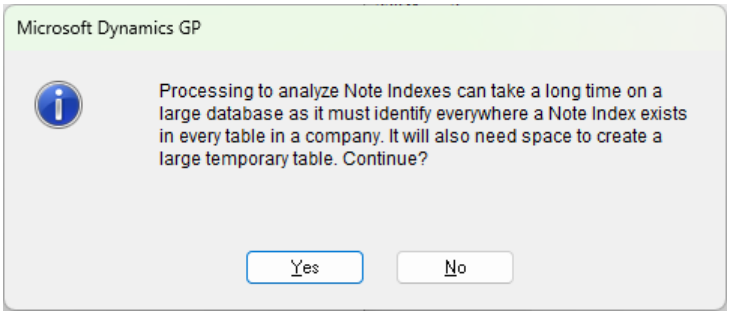
*Log Elapsed Times*

This checkbox (on by default) will write log entries to the GP Power Tools log for the current user and company with the Elapsed Times for each step of the Process to read the data and each step required to fix the Note Indexes.

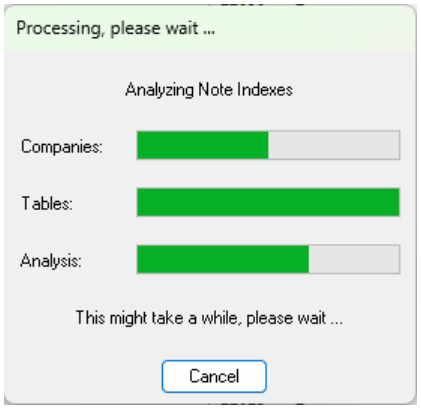
The following is a description of the additional buttons on the window:

*Process Button*

Use this button to process the selected companies to identify if there are Note Index issues that need to be resolved. This process can take some time, at least 10 minutes, on a large system.

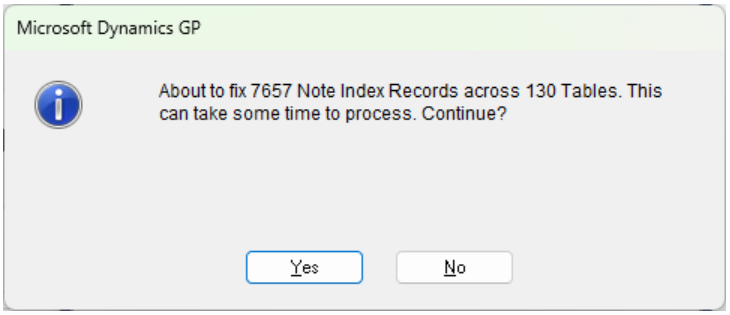



Once the user confirms to continue a processing window will be displayed. If background processing is available, the cancel button will be available to abort processing if desired.

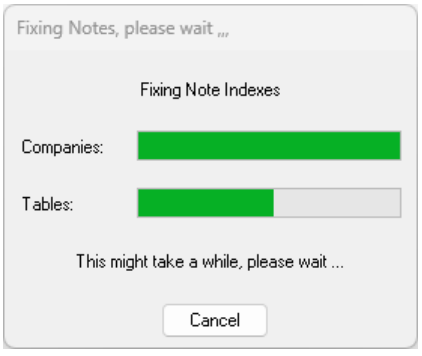



### *Fix Notes Button*

Use this button to fix the selected issues. When the button is clicked the number of records and tables to be fixed is displayed.



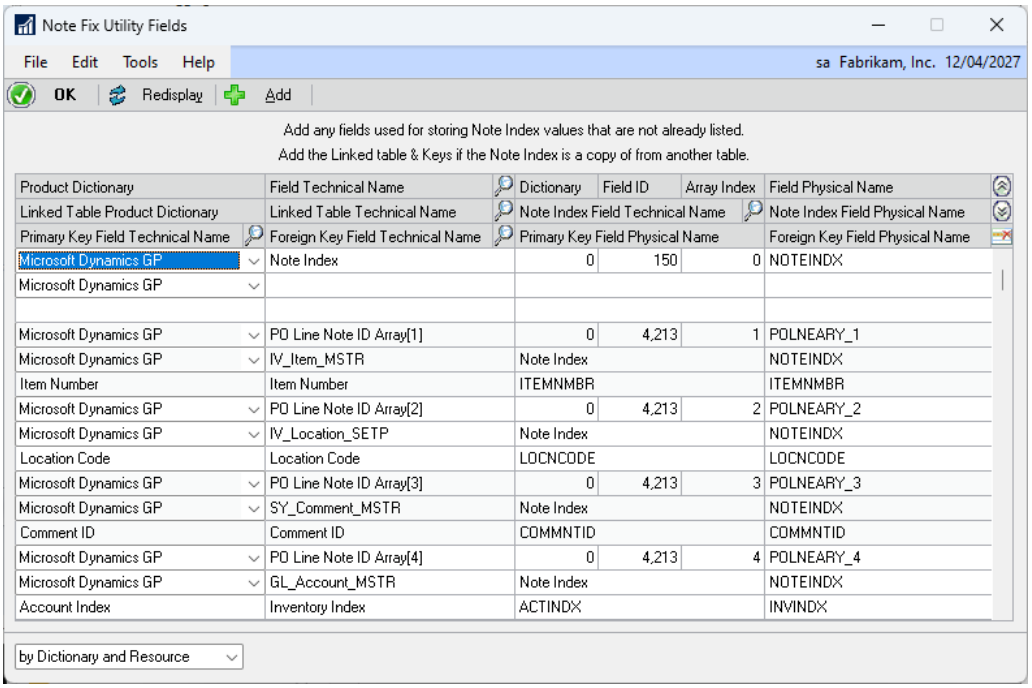

If Yes is selected there will be a final warning to ensure that databases are backed up and other users are logged out. Once the user confirms to continue a Fixing Note Indexes window will be displayed. If background processing is available, the cancel button will be available to abort processing if desired.




### *Edit Fields Button*

Use this button to open the Note Fix Utility Fields window which can be used to define the fields where Note Index values are stored.

*If the Note Index field is used to store a copy of a Note Index from another table, you can specify the other table and Note Index field in that table as well as the fields in both tables used to link the two tables together. Defining the Linked Table prevents the copied Note Indexes showing up as Duplicates and also ensures the copied value is updated if the original Note Index is changed.*



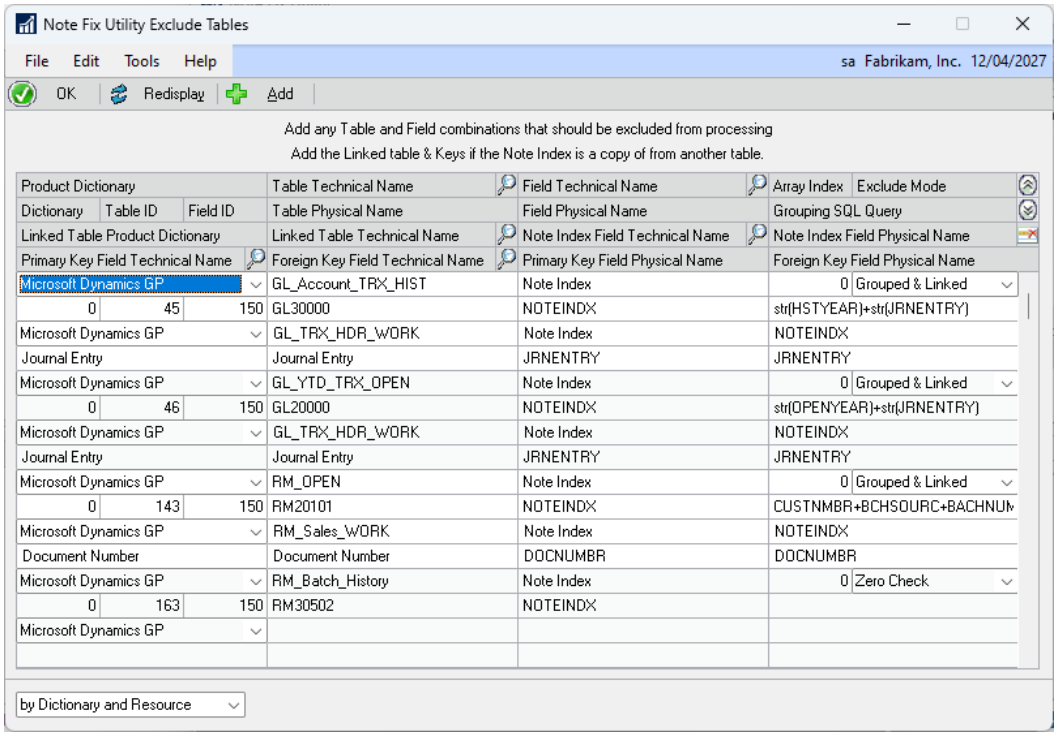

### *Exclude Tables Button*

Use this button to open the Note Fix Utility Tables window which can be used to exclude tables from processing, defined grouped tables, tables not used and define linked tables not linked already by the Fields already defined.

*If the Note Index field in a table is used to store a copy of a Note Index from another table, you can specify the other table and Note Index field in that table as well as the fields in both tables used to link the two tables together. Defining the Linked Table prevents the copied Note Indexes showing up as Duplicates and also ensures the copied value is updated if the original Note Index is changed.*

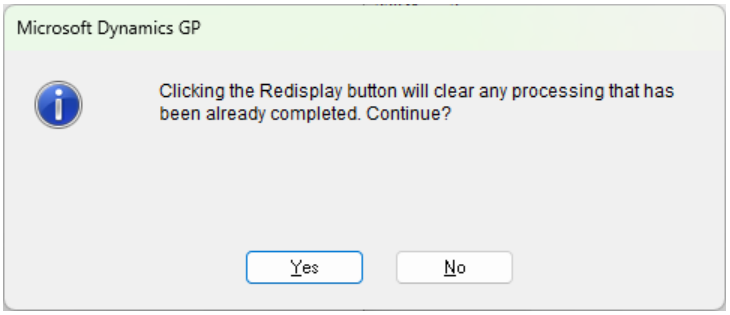
History tables can be excluded from the Zero Check as this is unnecessary to assign a Note Index value to a transaction that is no longer active. Table Fields that are currently not used in the application can be marked as Not Used and excluded from all checks.

*Grouped Tables can be used when a single Note Index is used on multiple table records because all table records refer to a single transaction. For example: the GL Transaction Open (GL20000) and GL Transaction History (GL30000) tables only store the journal lines and not the journal header, so all lines in a journal have the same Note Index. Using grouped tables prevents the multiple records being identified as duplicates and ensures that if the Note Index is updated, all the records in the group will be updated together.*

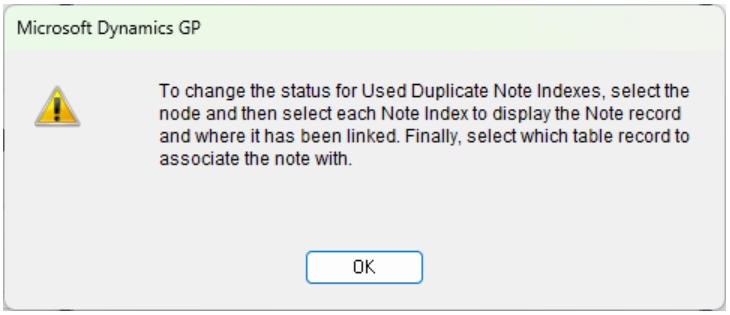



### *Redisplay Button*

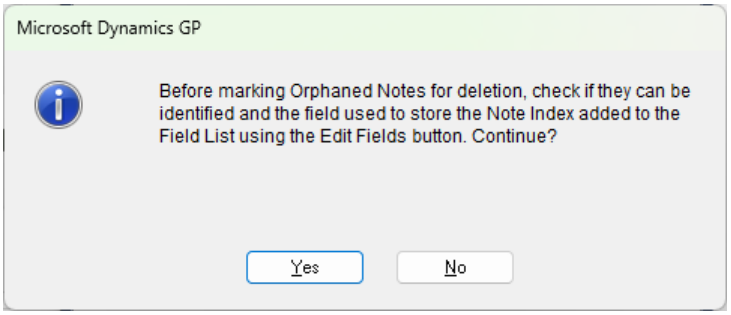
Use this button to reset the window so the companies can be processed again. A warning is displayed if you will be clearing the previously processed data to avoid accidental use of the button.




The following dialog is displayed if clicking on the Used Duplicate Note Indexes node in the Company Tree to explain that with a used note, the user must identify which is the correct record for that note. All other records using that Note Index will be assigned a new Note Index.

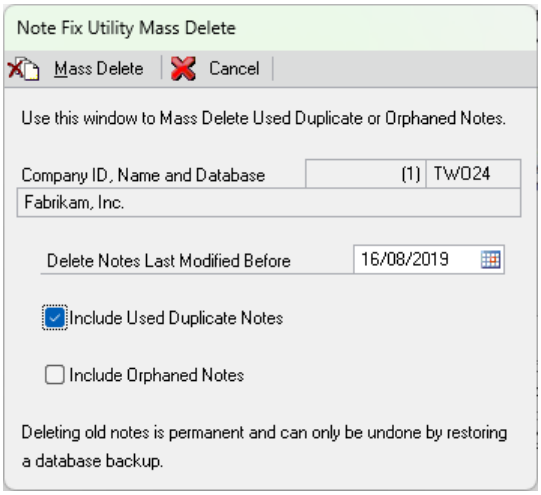


The following dialog is displayed if clicking on the Orphaned Note Indexes node in the Company Tree to explain that before you delete all the orphans, check if an ISV product is using a different field and update it using the Edit Fields Button.




### *Mass Delete Button*

Use this button to open the Note Fix Utility Mass Delete window, which can be used to delete note records older than a specified date. This can be a simpler solution to Duplicate and Orphaned Notes than trying to identify where the notes actually belong.



## Database Space Recovery

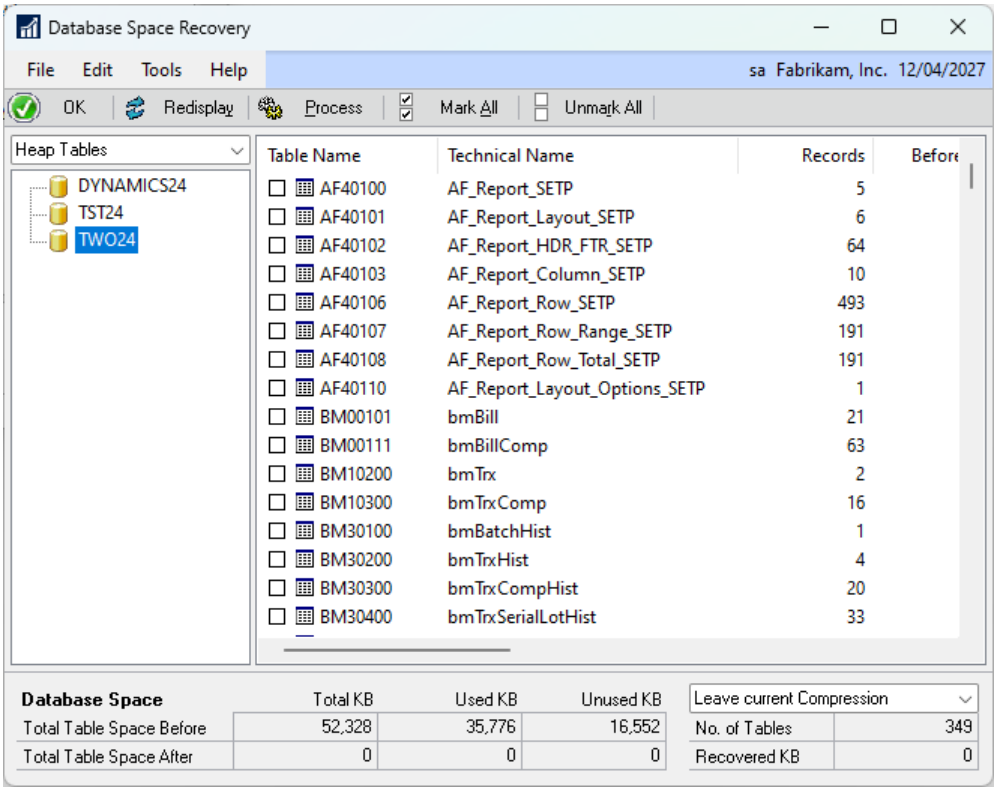
You can open the Database Space Recovery window by selecting Database Space Recovery from the Utilities section of the GP Power Tools Area Page or by selecting Database >> Database Space Recovery from the Options button drop list on the main window. This is an Advanced Mode feature.

The Database Space Recovery window is used to recover unused space from SQL Heap type tables and, on SQL Server installations that support compression, enable and disable Page Compression for SQL Heap type and Clustered tables.

*Tables in SQL Server can either be Clustered tables where one of the keys is clustered, or Heap Tables where there is no clustered index. Records in a Clustered table are stored in the order dictated by the clustered key and the table records are shuffled as records are added and deleted. Records in a Heap table are added at the end of the table when records are added and deleted records are marked as deleted, but the unused space is not removed.*

This tool removes the unused space created when records are deleted from Heap tables. It can also apply Page Compression to both Heap tables and Clustered tables on Enterprise and Azure SQL Server systems where compression is available.

Compressed tables are shown with a small red down arrow: 



The following is a description of the individual fields on the window:

*Table Type*

This drop down list selects the type of table for the window to operate on. The selection can be Heap Tables (tables without a clustered index) or Clustered Tables (tables with a clustered index). If SQL compression is not supported, this option will be disabled.

*Database Tree*

The left-hand pane displays the databases in the system.

*Table List*

The right-hand pane displays the tables with records for the selected type and database.

*Compression Mode*

This drop down list selects if SQL Page Compression for the tables should be left as is, enabled or disabled. If SQL compression is not supported, this option will be disabled.

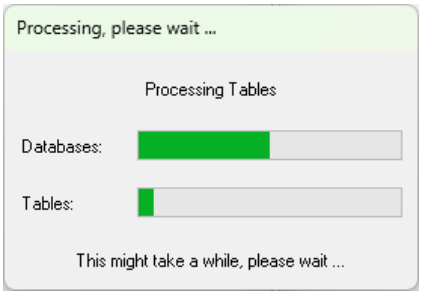
The following is a description of the additional buttons on the window:

*Redisplay Button*

Use this button to refresh the tables displayed and reset the processing status.

*Process Button*

Use this button to process the selected tables in each of the databases.




After processing, the table cannot be re-selected. Use the Redisplay Button to refresh the display and reset the table state. The statistics for the database will be updated to show the amount of space recovered:

| Database Space | Total KB | Used KB | Unused KB |
|---|---|---|---|
| Total Table Space Before | 61,336 | 54,144 | 7,192 |
| Total Table Space After | 27,160 | 18,432 | 8,728 |

Add Page Compression

| | |
|---|---|
| No. of Tables | 125 |
| Recovered KB | 34,176 |

*Mark All Button*

Use this button to mark all the tables (or all highlighted tables) to process.

*Unmark All Button*

Use this button to unmark all the tables (or all highlighted tables) to process.

*For SQL Server Enterprise or Azure systems that support compression, the decision to use compression or not and how it affects performance will depend on each system. Using Compression increases the CPU workload as there is an overhead to decompress the data after it has been read. If your Server is already CPU bound, compression would not be recommended (but an upgrade would be).*

*Overall performance can be improved with compression enabled, as the time to read uncompressed table data from the hard drive system can often still be longer than the time to read the compressed data and decompress it. CPU times is many factors faster than hard drive read and write times. The compressed data will also speed up backups as less space is required.*

*Microsoft Dynamics GP tables compress extremely well and it is not uncommon to see a 50% reduction is space. This is primarily because string fields in Dexterity are stored as the fixed length CHAR(X) datatype and padded with spaces. The first stage of SQL Compression will internally store all of these CHAR(X) columns as VARCHAR(X), meaning that all the unused spaces at the end of string fields, and the empty string fields, are removed from the stored data.*

# Additional Database Features

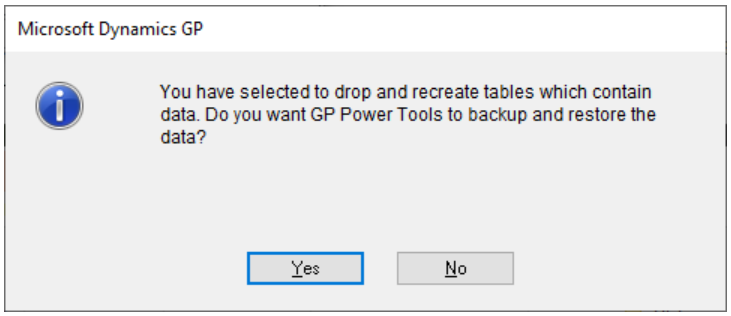
GP Power Tools adds some extra features to help database administrators. Below is a summary of the features:

## Send Password Reset Emails

When resetting passwords on the User Setup window, if the Password Reset Email is enabled, GP Power Tools will send an email to the user. This can be configured on the Password Reset Email Settings window available from the Additional menu.

## Keep Table Data for SQL Maintenance

When dropping and recreating tables using the SQL Maintenance window, GP Power Tools will offer to backup and restore table data if any of the selected tables contain data.




## Table Information for SQL Maintenance

GP Power Tools adds the table's Physical Name, Dexterity Technical Name and Number of records to the list of tables on the SQL Maintenance window.

# Chapter 8: Dex.ini Settings

## GP Power Tools Settings

GP Power Tools uses the Dex.ini file to store a number of settings. The default location for the Dex.ini file is in the data subfolder beneath the Microsoft Dynamics GP application folder. These settings are explained below:

### MBS_Debug_Path

This setting can point to a location for the Debugger.xml setup file. The default for this setting is missing, which means that the Debugger.xml file will be stored in the data subfolder beneath the Microsoft Dynamics GP application folder.

### MBS_Debug_SetupMode

This setting can be TRUE or missing and denotes whether Setup Mode is enabled. The default for this setting is missing, which means that Setup Mode is not enabled.

### MBS_Debug_AutoOpen

This setting can be TRUE or missing and denotes whether GP Power Tools window should open automatically after logging into a company.

### MBS_Debug_Version

This setting tracks the last used version of GP Power Tools on the current workstation.

### MBS_Debug_Install

This setting tracks the when GP Power Tools has been newly installed on the current workstation.

### WDC_InstallExclude

This setting tracks excluded launch files that will not be included on the Additional Launch File Installer dialog.

### MBS_Debug_LogOnStartup

This setting can be TRUE or missing and denotes whether to automatically start logging when Microsoft Dynamics GP is next started. The default for this setting is missing, which means that the feature is disabled.

### MBS_Debug_RuntimeCheck

This setting can be FALSE or missing and denotes whether the Runtime Engine version and build information is checked for compatibility. The default for this setting is missing, which means that the version and build will be checked.

### MBS_Debug_ShowRuntime

This setting can be TRUE or missing, and denotes whether the Runtime Engine is shown when creating Dexterity sanScript scripts in either the Trigger Setup window or the Runtime Execute Setup window. If this setting is enabled, the Resource Explorer window and Table Explorer window will also display resources from the Runtime Engine dictionary DEX.DIC. The default for this setting is missing, which means the runtime engine is not displayed.

### MBS_Debug_ConfigurationOverride

This setting can be TRUE or missing, and denotes whether GP Power Tools is allowed to automatically update Dex.ini Settings for this workstation as defined in the Dex.ini Configuration window. Set to TRUE to prevent any updates.

### MBS_Debug_LaunchConfigurationOverride

This setting can be TRUE or missing, and denotes whether GP Power Tools is allowed to automatically change the Launch File for this workstation as defined in the Launch File Configuration window. Set to TRUE to prevent any updates.

### MBS_Debug_LogAppDetails

This setting can be TRUE or missing and denotes whether GP Power Tools should log an entry into the GPPTools_<User>_<Company>.log file each time a user logs into a company.

### SQLLogRename

This setting can be used to automatically rename the DEXSQL.LOG file each day. The value will be the date of the last rename in the form YYYYMMDD.

### SQLLastCompany

This setting is used to automatically store the last Company ID selected for the current workstation. This allows the company selection drop-down list to be defaulted to the last company used.

### DefaultLastCompany

This setting can be FALSE or missing and is used to disable the automatic defaulting of the last company used when logging into Microsoft Dynamics GP or switching companies.

### MBS_Debug_UpdateLastUserOnExit

This setting can be FALSE or missing and is used to disable writing the last user and company details when exiting from Microsoft Dynamics GP.

### MBS_Debug_CompanySwitchWidth

This setting can be TRUE or missing and is used to expand the fields on the Company Login window to use the full width of the window.

### MBS_Debug_WinDebugger

This setting is used to store the last window size, position, and state for the GP Power Tools main window.

### MBS_Debug_WinDebuggerSetup
This setting is used to store the last window size, position, and state for the GP Power Tools Setup window.

### MBS_Debug_WinDebuggerStatus
This setting is used to store the last window size, position, and state for the Trigger Status window.

### MBS_Debug_WinResourceInformation
This setting is used to store the last window size, position, and state for the Resource Information window.

### MBS_Debug_WinResourceFinder
This setting is used to store the last window size, position, and state for the Resource Finder window.

### MBS_Debug_WinSecurityProfiler
This setting is used to store the last window size, position, and state for the Security Profiler window.

### MBS_Debug_WinSecurityInfo
This setting is used to store the last window size, position, and state for the Security Information window.

### MBS_Debug_WinSecurityInfoResource
This setting is used to store the last window size, position, and state for the Security Information Resources window.

### MBS_Debug_WinSecurityLog
This setting is used to store the last window size, position, and state for the Security Log window.

### MBS_Debug_WinSecurityLogDetail
This setting is used to store the last window size, position, and state for the Security Log Details window.

### MBS_Debug_WinSecurityLogResource
This setting is used to store the last window size, position, and state for the Security Log Resource Details window.

### MBS_Debug_WinSecurityAnalyzer
This setting is used to store the last window size, position, and state for the Security Analyzer window.

### MBS_Debug_WinSecurityEnhanced
This setting is used to store the last window size, position, and state for the Enhanced Security window.

### MBS_Debug_WinSecurityDeny
This setting is used to store the last window size, position, and state for the Security Denied window.

### MBS_Debug_WinSecurityHide
This setting is used to store the last window size, position, and state for the Security Hide window.

### MBS_Debug_WinDictionaryControl
This setting is used to store the last window size, position, and state for the Dictionary Control window.

### MBS_Debug_WinCompanyFilter
This setting is used to store the last window size, position, and state for the Company Login Filter window.

### MBS_Debug_WinWindowMemory
This setting is used to store the last window size, position, and state for the Window Position Memory window.

### MBS_Debug_WinActivityLog
This setting is used to store the last window size, position, and state for the User Activity Log window.

### MBS_Debug_WinActivityLogDetail
This setting is used to store the last window size, position, and state for the User Activity Log Detail window.

### MBS_Debug_WinActivityLogMaxUser
This setting is used to store the last window size, position, and state for the User Activity Log Maximum Users window.

### MBS_Debug_WinLoginLimits
This setting is used to store the last window size, position, and state for the Login Limits window.

### MBS_Debug_WinLaunchFileConfig
This setting is used to store the last window size, position, and state for the Launch File Configuration window.

### MBS_Debug_WinProductSelection
This setting is used to store the last window size, position, and state for the Dynamic Product Selection window.

### MBS_Debug_WinWebsiteSettings
This setting is used to store the last window size, position, and state for the Website Settings window.

### MBS_Debug_WinProductVersion
This setting is used to store the last window size, position, and state for the Product Version Validation window.

### MBS_Debug_WinXMLTableExport
This setting is used to store the last window size, position, and state for the XML Table Export window.

### MBS_Debug_WinXMLTableImport
This setting is used to store the last window size, position, and state for the XML Table Import window.

### MBS_Debug_WinDatabaseValidation
This setting is used to store the last window size, position, and state for the Database Validation window.

### MBS_Debug_WinLoginMaintenance
This setting is used to store the last window size, position, and state for the SQL Login Maintenance window.

### MBS_Debug_WinCopyUserSettings
This setting is used to store the last window size, position, and state for the Copy User Settings window.

### MBS_Debug_WinSQLTriggerControl
This setting is used to store the last window size, position, and state for the SQL Trigger Control window.

### MBS_Debug_WinNoteFixUtility
This setting is used to store the last window size, position, and state for the Note Fix Utility window.

### MBS_Debug_WinDatabaseSpaceRecovery
This setting is used to store the last window size, position, and state for the Database Space Recovery window.

### MBS_Debug_WinProjectSetup
This setting is used to store the last window size, position, and state for the Project Setup window.

### MBS_Debug_WinRuntimeExecute
This setting is used to store the last window size, position, and state for the Runtime Execute Setup window.

### MBS_Debug_WinRuntimeExecuter
This setting is used to store the last window size, position, and state for the Runtime Executer window.

### MBS_Debug_WinSQLExecute
This setting is used to store the last window size, position, and state for the SQL Execute Setup window.

### MBS_Debug_WinSQLExecuter
This setting is used to store the last window size, position, and state for the SQL Executer window.

### MBS_Debug_WinSQLResults
This setting is used to store the last window size, position, and state for the SQL Results window.

### MBS_Debug_WinNetExecute
This setting is used to store the last window size, position, and state for the Net Execute window.

### MBS_Debug_WinNetExecuter
This setting is used to store the last window size, position, and state for the Net Executer window.

### MBS_Debug_WinParameterMaintenance
This setting is used to store the last window size, position, and state for the Parameter List Maintenance window.

### MBS_Debug_WinMessagesSetup
This setting is used to store the last window size, position, and state for the Messages Setup window.

### MBS_Debug_WinSnippetSetup
This setting is used to store the last window size, position, and state for the Snippet Setup window.

### MBS_Debug_WinFormControlSetup
This setting is used to store the last window size, position, and state for the Form Control Setup window.

### MBS_Debug_WinPasswordSetup
This setting is used to store the last window size, position, and state for the Password Setup window.

### MBS_Debug_WinFormControlStatus
This setting is used to store the last window size, position, and state for the Form Control Status window.

### MBS_Debug_WinFormControlResources
This setting is used to store the last window size, position, and state for the Form Control Resources window.

### MBS_Debug_WinTriggerListMaintenance
This setting is used to store the last window size, position, and state for the Dynamic Trigger Logging window.

### MBS_Debug_WinConfigurationExportImport
This setting is used to store the last window size, position, and state for the Configuration Export/Import window.

### MBS_Debug_WinConfigurationMaintenance
This setting is used to store the last window size, position, and state for the Configuration Maintenance window.

### MBS_Debug_WinScreenShot
This setting is used to store the last window size, position and state for the ScreenShot window.

### MBS_Debug_WinLoggingSettings
This setting is used to store the last window size, position and state for the Logging Settings window.

### MBS_Debug_WinEmailSettings
This setting is used to store the last window size, position and state for the Email Settings window.

### MBS_Debug_WinAdminSettings
This setting is used to store the last window size, position and state for the Administrator Settings window.

### MBS_Debug_WinConfigSettings
This setting is used to store the last window size, position and state for the Dex.ini Configuration window.

### MBS_Debug_WinSendEmail
This setting is used to store the last window size, position and state for the Send Email window.

### MBS_Debug_WinResourceExplorer
This setting is used to store the last window size, position and state for the Form Explorer window.

### MBS_Debug_WinMenuExplorer
This setting is used to store the last window size, position and state for the Menu Explorer window.

### MBS_Debug_WinTableExplorer
This setting is used to store the last window size, position and state for the Table Explorer window.

### MBS_Debug_WinReportExplorer
This setting is used to store the last window size, position and state for the Report Explorer window.

### MBS_Debug_WinObjectExplorer
This setting is used to store the last window size, position and state for the Security Object Explorer window.

### MBS_Debug_WinScriptExplorer
This setting is used to store the last window size, position and state for the Script Explorer window.

### MBS_Debug_WinTableLookup
This setting is used to store the last window size, position and state for the Table Lookup window.

### MBS_Debug_WinFieldLookup
This setting is used to store the last window size, position and state for the Field Lookup window.

### MBS_Debug_WinGlobalExplorer
This setting is used to store the last window size, position and state for the Global Variables Explorer window.

### MBS_Debug_WinConstantExplorer
This setting is used to store the last window size, position and state for the Constant Explorer window.

### MBS_Debug_WinFormExplorer
This setting is used to store the last window size, position and state for the Form Explorer window.

### MBS_Debug_WinCalculator
This setting is used to store the last window size, position and state for the Calculator window.

### MBS_Debug_WinKeyLookup
This setting is used to store the last window size, position and state for the Table Keys Lookup window.

### MBS_Debug_WinDAGControl
This setting is used to store the last window size, position and state for the Dictionary Assembly Generator Control window.

### MBS_Debug_WinScreenOutput
This setting is used to store the last window size, position and state for the Report Writer Screen Output window.

### MBS_Debug_DisableScreenOutputMemory
This setting can be used to disable the window position memory feature for the Report Writer Screen Output window..

### MBS_Debug_Automate_File
This setting is used by Microsoft Support to provide the full path or filename to a Diagnostics configuration settings file to be loaded after logging into Microsoft Dynamics GP. If the full path is not provided, the file can be located in the Debugger logs folder, the application's Data folder, or the folders where the DYNAMICS.EXE or DYNAMICS.SET are located. Trigger IDs, Script IDs and Profile IDs loaded with this option should be prefixed with a tilde (~) character. By default, this setting is removed after use.

### MBS_Debug_Automate_Script
This setting is used by Microsoft Support to provide the Script ID for a Runtime Execute Setup Diagnostics script to be executed after logging into Microsoft Dynamics GP. The Script ID executed with this option should be prefixed with a tilde (~) character. By default, this setting is removed after use.

### MBS_Debug_Automate_Status

This setting is used by Microsoft Support to control to the behavior of the Diagnostics automation features of GP Power Tools. By default, this setting is removed after use. The valid flags (which can be added together) are as follows:

1 – Do not delete settings loaded from configuration settings file.
2 – Do not delete Diagnostics Automation Dex.ini settings.
4 – Do not delete configuration settings XML file.
8 – Do not display "Please Wait" dialogs while loading settings file.

### MBS_Debug_DisableSplitters

This setting can be used disable the splitter functionality on the Security Information and Resource Explorer windows. Set it to TRUE to disable the splitters.

### MBS_Debug_VBADisableReset

This setting is used by GP Power Tools to signify that Visual Basic for Applications (VBA) should be re-enabled after one login.

### MBS_Debug_VSTDisable

This setting is used by GP Power Tools to disable Visual Studio Tools Addins on login.

### MBS_Debug_VSTDisableReset

This setting is used by GP Power Tools to signify that Visual Studio Tools Addins should be re-enabled after one login.

### MBS_Debug_SkipVersionChecks

This setting is used to allow GP Power Tools to run on a different version of Dexterity than the one it was built for. It is to be used when testing GP Power Tools on upcoming versions of Microsoft Dynamics GP.

### MBS_Debug_LastRunSystem

This setting is used to track when GP Power Tools was last run on a particular workstation.

### MBS_Debug_LastRunUser

This setting is used to track when GP Power Tools was last run by a particular user.

### MBS_Debug_LogWinData

This setting is used to enable logging for the automatic window positioning code for troubleshooting purposes.

### MBS_Debug_CompanyFilter

This setting is used to specify the Company Login Filter Profile ID to use for the current workstation.

### MBS_Debug_LogListPath

This setting is used to specify the text file containing the settings for Dynamic Trigger Logging.

### MBS_Debug_Break

This setting can be TRUE or missing and is used to force the Script Debugger (if enabled) to open automatically when starting Microsoft Dynamics GP.

### MBS_Debug_LookupPosition

This setting can be FALSE or missing and is used to disable the Lookup Window Positioning which ensures Lookup windows open next to the calling window.

### MBS_Debug_NamesUseClipboard

This setting can be TRUE or missing and is used to enable Names Button Uses Clipboard option on the script menu.

### MBS_Debug_CaptureSettings

This setting can be TRUE or missing and is used to enable the logging of reads to Dex.ini settings which do not exist in the Dex.ini file. The data captured can be printed from the Dex.ini Configuration window.

### MBS_Debug_DisableWebsiteSettings

This setting can be TRUE or missing and is used to disable changing of the default websites for the Connect and Intelligent Cloud Insights (GP 2018 R2 or later) homepage sections. The settings are controlled on the Website Settings window.

### MBS_Debug_ProductVersionOverride

This setting can be TRUE or missing and is used to disable displaying of the Product Version Validation warning dialog on a workstation.

### MBS_Debug_WCBackground

This setting can be TRUE or missing and is used to force background processing during login on the Web Client. It was used when debugging an issue when the Web Client would not initialize the home page after login.

### MBS_Debug_ValidateLaunchFile

This setting can be FALSE or missing and is used to disable validation of folder paths used in the launch file during login. Having non-existent paths referenced in the launch file can cause issues with reading version numbers from dictionaries.

### MBS_Debug_DexIniCheck

This setting can be FALSE or missing and is used to disable checking the system Dex.ini file can be written to. Making the Dex.ini file read-only can cause issues with products that write settings to the Dex.ini file.

### MBS_Debug_ExportCompatibilityWarning

This setting can be FALSE or missing and is used to disable the displaying of compatibility warnings when exporting projects from Project Setup, if the project will not be compatible with Build 28 or earlier.

### DefaultLastUserWindows

This setting can be used to enable the defaulting of the User ID to the current Windows User ID when logging into Microsoft Dynamics GP.

### SuppressChangeDateForce

This setting can be used to force the User Date to change at midnight even when the date change request dialog is suppressed. Use in conjunction with the SuppressChangeDateDialog setting.

### MBS_Debug_DisableTimedProcessRestore

This setting can be disable GP Power Tools applying a fix for the bug in the Dexterity function Activity_GetBackgroundStatus() where it removes timed processes when called.

### MBS_Debug_HideGames

This setting can be used to hide the games from the Microsoft Dynamics GP >> Tools >> Customize menu.

### MBS_Debug_UserMessageReplace

This setting can be set to FALSE to disable the replacement of the User Message dialogs which an Automatic Logout friendly custom form.

## System Settings

GP Power Tools can also manipulate the values of certain system settings stored in the Dex.ini settings file:

### SQLLogSQLStmt

This setting can be TRUE or FALSE and controls whether statements Microsoft Dynamics GP sends to the SQL Server are logged to the DEXSQL.LOG file by default.

### SQLLogODBCMessages

This setting can be TRUE or FALSE and controls whether ODBC messages returned from the SQL Server back to the Microsoft Dynamics GP client are logged to the DEXSQL.LOG file by default.

### SQLLogAllODBCMessages

This setting can be TRUE or FALSE and controls whether all ODBC messages returned from the SQL Server back to the Microsoft Dynamics GP client are logged to the DEXSQL.LOG file by default.

### SQLLogPath

This setting can be used to change the default location of the DEXSQL.LOG file.

### ScriptDebugger

This setting can be TRUE or FALSE and controls whether the Dexterity Debug menu is available in runtime mode.

### ScriptDebuggerProduct

This setting contains the Dexterity Product ID that will be used to set the initial context of the Debug menu. The default value is 0 for Dynamics.

### ShowDebugMessages

This setting can be TRUE or FALSE and controls whether internal debug message dialogs are displayed when the Debug Menu is enabled. It is recommended that this should be set to FALSE for production systems.

### ScriptLogEnhanced

This setting can be TRUE or FALSE and controls whether the enhanced Dexterity Script Log features are enabled. Enabling this option adds time stamps and flagging of background processes to the script log. The default value is set to TRUE by GP Power Tools.

### ApplicationName

This setting contains the name to be shown on the title bar when the application first launches. If this value is not defined, the name in the title bar will default to “Dexterity Runtime”.

### AutoInstallChunks

This setting allows chunks to be included without prompting when Microsoft Dynamics GP is launched.

### AllowWrongDex
This setting allows a mismatched Dex.dic and Dexterity Runtime version to be used. It is not recommended to use this option.

### SkipVersionChecks
This setting allows Microsoft Dynamics GP to launch without errors even when the dictionary version numbers do not match the version information in the database. It is not recommended to use this option.

### SAMPLEDATEMSG
This setting prevents the Fabrikam sample company date warning dialog from opening when logging in.

### SQLLoginCompatibilityMode
This setting controls if Microsoft Dynamics GP continues to use SQL Login Compatibility Mode.

### ExportOneLineBody
This setting controls whether text report body sections are exported as a single line in the export file.

### ExportLinesPerPage
This setting controls the number of lines to include on a report page when it is exported to a file.

### ExportPDFLinesPerPage
This setting controls the number of lines to include on a report page when it is exported to a PDF file.

### DebugRW
This setting is used to configure the Report Writer to create a debugging log file named DebugRW.txt that will appear in the data subfolder beneath the Microsoft Dynamics GP application folder.

### SuppressChangeDateDialog
This setting prevents the Change Date dialog from being displayed at midnight. If used, the User Date will not change at midnight. Use the SuppressChangeDateForce setting to force the date to change.

### ShowAdvancedMacroMenu
This setting will enable the Advanced Macro Menu from the Tools >> Macro menu.

### ShowAllMenuItems
This setting will leave all menu items showing even if the module is not installed or if access is denied.

### SuppressSound
This setting disables all sound from the Microsoft Dynamics GP application.

### QueueMoreInfo
This setting can be used to enable the More Info button on the Process Monitor window.

### MouseWheel
This setting can be used to disable Mouse Wheel scrolling in the application.

### MaxSWScrollbarSize
This setting can be used to override the width of scrollbars in the application. The default value is 17 pixels.

### DebugFonts
This setting can be used to enable logging of Report Writer selections to the DebugLog.txt file.

### TPELogging
This setting can be used to enable logging of the internals of the Template Processing Engine (TPE) for word templates.

### KeepTemplateTempFiles
This setting can be used to disable the automatic removal of the temporary files used when the Template Processing Engine (TPE) runs.

### VBADisable
This setting can be used to disable Visual Basic for Applications when restarting Microsoft Dynamics GP.

### EnableServerDropDown
This setting can be used to disable the Data Source Server selection when logging into Microsoft Dynamics GP.

### DefaultLastUser
This setting can be used to disable the defaulting of the last user used when logging into Microsoft Dynamics GP.

### EnableWCRibbons
This setting can be used to disable the GP 2013 R2 or later Web Client style ribbons in the desktop client for the current workstation.

### WindowMax
This setting can be used to control whether the application opens full screen for the current workstation.

### WindowPosX & WindowPosY
These settings can be used to control the default application position when not maximized for the current workstation.

### WindowWidth & WindowHeight
These settings can be used to control the default application size when not maximized for the current workstation.

### OLEClose

This setting can be used to control whether the application attempts to close the OLE Contain.exe program on exit for the current workstation

## Script Editor Settings

GP Power Tools uses some of the Dexterity Script Editor Dex.ini settings:

### ScriptEditorSyntaxColoring
This setting stores whether Syntax Highlighting is enabled.

### ScriptKeywordColor
This setting stores the color selection for keywords.

### ScriptIdentifierColor
This setting stores the color selection for identifiers.

### ScriptNumberColor
This setting stores the color selection for numbers.

### ScriptStringColor
This setting stores the color selection for strings.

### ScriptCommentColor
This setting stores the color selection for comments.

### ScriptOperatorColor
This setting stores the color selection for operators.

### ScriptErrorColor
This setting stores the color selection to display Scripting Highlighting errors.

### ScriptEditorFontName
This setting stores the font style section.

### ScriptEditorFontSize
This setting stores the font size section.

# Chapter 9: Helper Functions

GP Power Tools has many helper functions which can be used to make cross-dictionary Dexterity sanScript simpler to write. The Helper Function Assistant window will automatically insert the code required to use these functions.

Below are the details of the helpers available:

- *MBS_Get_Window_Value*
- *MBS_Get_Window_Value_Boolean*
- *MBS_Get_Window_Value_Date*
- *MBS_Get_Window_Value_Numeric*
- *MBS_Get_Window_Value_String*
- *MBS_Get_Window_Value_Text*
- *MBS_Get_Window_Value_Time*
- *MBS_Get_Window_Value_Exists*
- *MBS_Get_Window_Value_Modified*
- *MBS_Get_Window_Value_Modified_Boolean*
- *MBS_Get_Window_Value_Modified_Date*
- *MBS_Get_Window_Value_Modified_Numeric*
- *MBS_Get_Window_Value_Modified_String*
- *MBS_Get_Window_Value_Modified_Text*
- *MBS_Get_Window_Value_Modified_Time*
- *MBS_Get_Window_Value_Modified_Exists*

- *MBS_Set_Window_Value*
- *MBS_Set_Window_Value_Boolean*
- *MBS_Set_Window_Value_Date*
- *MBS_Set_Window_Value_Numeric*
- *MBS_Set_Window_Value_String*
- *MBS_Set_Window_Value_Text*
- *MBS_Set_Window_Value_Time*
- *MBS_Set_Window_Value_Focus*
- *MBS_Set_Window_Value_Focus_Immediate*
- *MBS_Set_Window_Value_Enabled*
- *MBS_Set_Window_Value_ReadOnly*
- *MBS_Set_Window_Value_Visible*
- *MBS_Set_Window_Value_Modified*
- *MBS_Set_Window_Value_Modified_Boolean*
- *MBS_Set_Window_Value_Modified_Date*
- *MBS_Set_Window_Value_Modified_Numeric*
- *MBS_Set_Window_Value_Modified_String*
- *MBS_Set_Window_Value_Modified_Text*
- *MBS_Set_Window_Value_Modified_Time*
- *MBS_Set_Window_Value_Modified_Focus*
- *MBS_Set_Window_Value_Modified_Focus_Immediate*
- *MBS_Set_Window_Value_Modified_Enabled*
- *MBS_Set_Window_Value_Modified_ReadOnly*
- *MBS_Set_Window_Value_Modified_Visible*

- *MBS_Run_Window_Value*
- *MBS_Run_Window_Value_Modified*
- *MBS_Pull_Window_Focus*

- *MBS_Get_Table_Value1*
- *MBS_Set_Table_Value1*
- *MBS_Get_Table_Value2*
- *MBS_Set_Table_Value2*
- *MBS_Get_Table_Value3*
- *MBS_Set_Table_Value3*
- *MBS_Get_Table_Value4*
- *MBS_Set_Table_Value4*
- *MBS_Get_Table_Buffer_Value*
- *MBS_Get_Table_Buffer_Value_Boolean*
- *MBS_Get_Table_Buffer_Value_Date*
- *MBS_Get_Table_Buffer_Value_Numeric*
- *MBS_Get_Table_Buffer_Value_String*
- *MBS_Get_Table_Buffer_Value_Text*
- *MBS_Get_Table_Buffer_Value_Time*
- *MBS_Set_Table_Buffer_Value*
- *MBS_Set_Table_Buffer_Value_Boolean*
- *MBS_Set_Table_Buffer_Value_Date*
- *MBS_Set_Table_Buffer_Value_Numeric*
- *MBS_Set_Table_Buffer_Value_String*
- *MBS_Set_Table_Buffer_Value_Text*
- *MBS_Set_Table_Buffer_Value_Time*

- *MBS_Copy_To_Window*
- *MBS_Copy_From_Window*
- *MBS_Copy_To_Window_Modified*
- *MBS_Copy_From_Window_Modified*
- *MBS_Table_Buffer_Get*
- *MBS_Table_Buffer_Save*
- *MBS_Table_Buffer_Remove*
- *MBS_Table_Buffer_Release*
- *MBS_Table_Buffer_Range*
- *MBS_Table_Buffer_Clear*
- *MBS_Table_Buffer_Fill*

- *MBS_Runtime_Execute*
- *MBS_Runtime_Execute_Background*
- *MBS_Runtime_Execute_Delayed*
- *MBS_Runtime_Execute_After_Background*
- *MBS_Runtime_Execute_Modified*
- *MBS_Runtime_Execute_Modified_Background*
- *MBS_Runtime_Execute_Modified_Delayed*
- *MBS_Runtime_Execute_Modified_After_Background*
- *MBS_SQL_Set_Database*
- *MBS_SQL_Check_Exists*
- *MBS_SQL_Execute*
- *MBS_SQL_Get_Data*
- *MBS_SQL_Parse_Data*
- *MBS_SQL_Parse_Data_Boolean*
- *MBS_SQL_Parse_Data_Currency*
- *MBS_SQL_Parse_Data_Date*
- *MBS_SQL_Parse_Data_Datetime*
- *MBS_SQL_Parse_Data_Integer*
- *MBS_SQL_Parse_Data_Long*
- *MBS_SQL_Parse_Data_String*
- *MBS_SQL_Parse_Data_Text*

- *MBS_SQL_Parse_Data_Time*
- *MBS_SQL_Parse_Data_VCurrency*
- *MBS_SQL_Parse_Data_Reset*
- *MBS_Export_SQL_Query_To_File*
- *MBS_SQL_Results*
- *MBS_SQL_Results_Immediate*
- *MBS_SQL_Results_Goto*
- *MBS_SQL_Results_Immediate_Goto*
- *MBS_SQL_Results_Close*
- *MBS_SQL_Results2*
- *MBS_SQL_Results_Immediate2*
- *MBS_SQL_Results_Goto2*
- *MBS_SQL_Results_Immediate_Goto2*
- *MBS_SQL_Results_Close2*
- *MBS_SQL_Goto_Get_Data*
- *MBS_SQL_Goto_Close*
- *MBS_SQL_Sort_Get*
- *MBS_SQL_Sort_Get*
- *MBS_SQL_Export_Data*
- *MBS_Net_Execute*
- *MBS_Script_Load_Dex*
- *MBS_Script_Load_SQL*
- *MBS_Script_Load_SQL_DB*
- *MBS_Script_Load_Net*

- *MBS_Param_Set*
- *MBS_Param_Get*
- *MBS_Param_Del*
- *MBS_Param_DelAll*

- *MBS_Memory_Set*
- *MBS_Memory_Set_Boolean*
- *MBS_Memory_Set_Currency*
- *MBS_Memory_Set_Date*
- *MBS_Memory_Set_Long*
- *MBS_Memory_Set_String*
- *MBS_Memory_Set_Time*
- *MBS_Memory_Set_Reference*
- *MBS_Memory_Set_Table*
- *MBS_Memory_Set_Field*
- *MBS_Memory_Get*
- *MBS_Memory_Get_Boolean*
- *MBS_Memory_Get_Currency*
- *MBS_Memory_Get_Date*
- *MBS_Memory_Get_Long*
- *MBS_Memory_Get_String*
- *MBS_Memory_Get_Time*
- *MBS_Memory_Get_Reference*
- *MBS_Memory_Del*
- *MBS_Memory_Del_Boolean*
- *MBS_Memory_Del_Currency*
- *MBS_Memory_Del_Date*
- *MBS_Memory_Del_Long*
- *MBS_Memory_Del_String*
- *MBS_Memory_Del_Time*
- *MBS_Memory_Del_Reference*

- *MBS_Get_Constant*
- *MBS_Get_Constant_Currency*
- *MBS_Get_Constant_Integer*
- *MBS_Get_Constant_String*
- *MBS_Set_Global*
- *MBS_Set_Global_Boolean*
- *MBS_Set_Global_Date*
- *MBS_Set_Global_Numeric*
- *MBS_Set_Global_String*
- *MBS_Set_Global_Text*
- *MBS_Set_Global_Time*
- *MBS_Get_Global*
- *MBS_Get_Global_Boolean*
- *MBS_Get_Global_Date*
- *MBS_Get_Global_Numeric*
- *MBS_Get_Global_String*
- *MBS_Get_Global_Text*
- *MBS_Get_Global_Time*

- *MBS_Auto_Log*
- *MBS_Logging_Start*
- *MBS_Logging_Stop*
- *MBS_Trigger_Start*
- *MBS_Trigger_Stop*
- *MBS_Trigger_Update_Dialog*
- *MBS_Trigger_Update_Email*
- *MBS_Arguments_Get_Count*
- *MBS_Arguments_Get_Type*
- *MBS_Arguments_Get_Value*
- *MBS_Arguments_Set_Value*

- *MBS_DUOS_Set*
- *MBS_DUOS_Get*
- *MBS_DUOS_Del*
- *MBS_DUOS_DelAll*
- *MBS_UserAddInfo_Get*
- *MBS_UserAddInfo_Set*
- *MBS_UserAddInfo_GetPrompt*
- *MBS_SQL_Lookup*
- *MBS_SQL_Lookup2*
- *MBS_SQL_Lookup_Parameter*
- *MBS_SQL_Lookup_Parameter2*
- *MBS_SQL_Lookup_Validate*
- *MBS_SQL_Lookup_Parameter_Validate*
- *MBS_Form_Lookup*
- *MBS_Form_Lookup2*
- *MBS_Form_Lookup_Parameter*
- *MBS_Form_Lookup_Parameter2*
- *MBS_Project_Start*
- *MBS_Project_Stop*

- *MBS_Script_Substitute*
- *MBS_Parameter_Placeholder*
- *MBS_Parameter_String*
- *MBS_Parameter_Number*
- *MBS_Parameter_Currency*

- *MBS_Parameter_Boolean*
- *MBS_Parameter_Date*
- *MBS_Parameter_Time*
- *MBS_Parameter_Load*
- *MBS_Parameter_Open*
- *MBS_Parameter_Set_String*
- *MBS_Parameter_Set_Number*
- *MBS_Parameter_Set_Currency*
- *MBS_Parameter_Set_Boolean*
- *MBS_Parameter_Set_Date*
- *MBS_Parameter_Set_Time*
- *MBS_Parameter_Get_String*
- *MBS_Parameter_Get_Number*
- *MBS_Parameter_Get_Currency*
- *MBS_Parameter_Get_Boolean*
- *MBS_Parameter_Get_Date*
- *MBS_Parameter_Get_Time*

- *MBS_Convert*
- *MBS_Convert_Boolean*
- *MBS_Convert_Currency*
- *MBS_Convert_Date*
- *MBS_Convert_Datetime*
- *MBS_Convert_Integer*
- *MBS_Convert_Long*
- *MBS_Convert_String*
- *MBS_Convert_Text*
- *MBS_Convert_Time*
- *MBS_Convert_VCurrency*

- *MBS_Return_By_Field*
- *MBS_Return_By_Field2*
- *MBS_Return_By_Reference*
- *MBS_Return_By_Reference2*
- *MBS_Map_By_Field*
- *MBS_Map_By_Reference*
- *MBS_Map*
- *MBS_Map_Boolean*
- *MBS_Map_Date*
- *MBS_Map_Numeric*
- *MBS_Map_String*
- *MBS_Map_Text*
- *MBS_Map_Time*

- *MBS_Get_Message*
- *MBS_Get_Message_Prompts*
- *MBS_getmsg*
- *MBS_Get_Error_Message*
- *MBS_Show_Dialog*
- *MBS_Show_Dialog_Text*
- *MBS_Ask_Dialog*
- *MBS_Ask_Dialog_Text*
- *MBS_Get_DateTime*
- *MBS_Token*
- *MBS_Field_ParseText*
- *MBS_subtext*

- *MBS_Security_Form_Check*

- *MBS_Trigger_Disable*
- *MBS_Trigger_Enable*
- *MBS_Trigger_DisableSingle*
- *MBS_Trigger_EnableSingle*
- *MBS_Is_Trigger_Started*
- *MBS_Is_Trigger_Enabled*

- *MBS_Exit_After_Processes*
- *MBS_Switch_Company*
- *MBS_CompanyColorGetRGB*
- *MBS_Copy_To_Clipboard*
- *MBS_Copy_From_Clipboard*
- *MBS_Show_Desktop_Alert*
- *MBS_Email_API*

- *MBS_Add_Virtual_Field*
- *MBS_Add_Virtual_FieldPrompt*
- *MBS_Add_Virtual_FieldFormat*
- *MBS_Add_Virtual_FieldPromptLookup*
- *MBS_Add_Virtual_FieldPromptFormat*
- *MBS_Add_Virtual_FieldAll*
- *MBS_Add_Virtual_FieldLine*
- *MBS_Expand_Virtual_Field_Window*
- *MBS_Get_Field_Reference*
- *MBS_Get_Virtual_Field*
- *MBS_Set_Virtual_Field*
- *MBS_Map_Virtual_Field*
- *MBS_Get_Virtual_Field_Caption*
- *MBS_Set_Virtual_Field_Caption*
- *MBS_Get_Virtual_Field_Tooltip*
- *MBS_Set_Virtual_Field_Tooltip*

- *MBS_Ask_Password*
- *MBS_Control_Start*
- *MBS_Control_Stop*
- *MBS_Control_Stop_All*
- *MBS_Control_Update_Dialog*
- *MBS_Control_Update_Expression*
- *MBS_Get_First_Window*
- *MBS_Check_Resource_Exists*

## MBS_Get_Window_Value

This call is used to obtain the value of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
call with name "MBS_Get_Window_Value" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Boolean

This call is used to obtain the value of a boolean window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out boolean OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;
call with name "MBS_Get_Window_Value_Boolean" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Date

This call is used to obtain the value of a date window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out date OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;
call with name "MBS_Get_Window_Value_Date" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Numeric

This call is used to obtain the value of a numeric window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out vcurrency OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;
call with name "MBS_Get_Window_Value_Numeric" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_String

This call is used to obtain the value of a string window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out string OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
call with name "MBS_Get_Window_Value_String" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Text

This call is used to obtain the value of a text window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out text OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;
call with name "MBS_Get_Window_Value_Text" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Time

This call is used to obtain the value of a time window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out time OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;
call with name "MBS_Get_Window_Value_Time" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Exists

This call is used to check the existence of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Exists;
```

An example script is:

```
local integer l_exists;
call with name "MBS_Get_Window_Value_Exists" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_exists;
if l_exists = OKAY then
    warning str(l_exists);
end if;
```

## MBS_Get_Window_Value_Modified

This call is used to obtain the value of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
call with name "MBS_Get_Window_Value_Modified" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Boolean

This call is used to obtain the value of a boolean modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out boolean OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;
call with name "MBS_Get_Window_Value_Modified_Boolean" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Date

This call is used to obtain the value of a date modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out date OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;
call with name "MBS_Get_Window_Value_Modified_Date" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Numeric

This call is used to obtain the value of a numeric modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out vcurrency OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;
call with name "MBS_Get_Window_Value_Modified_Numeric" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_String

This call is used to obtain the value of a string modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out string OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
call with name "MBS_Get_Window_Value_Modified_String" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Text

This call is used to obtain the value of a text modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out text OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;
call with name "MBS_Get_Window_Value_Modified_Text" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Time

This call is used to obtain the value of a time modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out time OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;
call with name "MBS_Get_Window_Value_Modified_Time" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Window_Value_Modified_Exists

This call is used to check the existence of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Exists;
```

An example script is:

```
local integer l_exists;
call with name "MBS_Get_Window_Value_Modified_Exists" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_exists;
if l_exists = OKAY then
    warning str(l_exists);
end if;
```

## MBS_Set_Window_Value

This call is used to set the value of a window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Boolean

This call is used to set the value of a boolean window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;
l_field = false;
call with name "MBS_Set_Window_Value_Boolean" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Date

This call is used to set the value of a date window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in date IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;
l_field = mkdate(1, 1, 1980);
call with name "MBS_Set_Window_Value_Date" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Numeric

This call is used to set the value of a numeric window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in vcurrency IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;
l_field = 0;
call with name "MBS_Set_Window_Value_Numeric" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_String

This call is used to set the value of a string window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in string IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value_String" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Text

This call is used to set the value of a text window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in text IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value_Text" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Time

This call is used to set the value of a time window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in time IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;
l_field = mktime(0, 0, 0);
call with name "MBS_Set_Window_Value_Time" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Focus

This call is used to set the focus to a window field from any open form in any dictionary. The focus change takes effect when all other scripts has been completed.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Focus" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Focus_Immediate

This call is used to set the focus to a window field from any open form in any dictionary. The focus change takes effect immediately.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Focus_Immediate" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Enabled

This call is used to set the enabled/disabled state of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Enabled;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Enabled" in dictionary 5261,
Dictionary, "Form", "Window", "Field", true, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_ReadOnly

This call is used to set the locked/unlocked state of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_ReadOnly;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_ReadOnly" in dictionary 5261,
Dictionary, "Form", "Window", "Field", true, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Visible

This call is used to set the hidden/shown state of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Visible;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Visible" in dictionary 5261,
Dictionary, "Form", "Window", "Field", true, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified

This call is used to set the value of a modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value_Modified" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_field, true {run script},
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Boolean

This call is used to set the value of a boolean modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;
l_field = false;
call with name "MBS_Set_Window_Value_Modified_Boolean" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Date

This call is used to set the value of a date modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in date IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;
l_field = mkdate(1, 1, 1980);
call with name "MBS_Set_Window_Value_Modified_Date" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Numeric

This call is used to set the value of a numeric modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in vcurrency IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;
l_field = 0;
call with name "MBS_Set_Window_Value_Modified_Numeric" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_String

This call is used to set the value of a string modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in string IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value_Modified_String" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Text

This call is used to set the value of a text modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in text IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;
l_field = "Value";
call with name "MBS_Set_Window_Value_Modified_Text" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Time

This call is used to set the value of a time modified window field from any open form in any dictionary. You have the option to also run the target field's change script.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in time IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;
l_field = mktime(0, 0, 0);
call with name "MBS_Set_Window_Value_Modified_Time" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_field, true {run
script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Focus

This call is used to set the focus to a modified window field from any open form in any modified dictionary. The focus change takes effect when all other scripts has been completed.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Modified_Focus" in dictionary
5261, Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Focus_Immediate

This call is used to set the focus to a modified window field from any open form in any modified dictionary. The focus change takes effect immediately.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Modified_Focus_Immediate" in
dictionary 5261, Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Enabled

This call is used to set the enabled/disabled state of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Enabled;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Modified_Enabled" in dictionary
5261, Dictionary, "Form", "Window", "Field", true, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_ReadOnly

This call is used to set the locked/unlocked state of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_ReadOnly;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Modified_ReadOnly" in
dictionary 5261, Dictionary, "Form", "Window", "Field", true,
l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Window_Value_Modified_Visible

This call is used to set the hidden/shown state of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Visible;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Window_Value_Modified_Visible" in dictionary
5261, Dictionary, "Form", "Window", "Field", true, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Run_Window_Value

This call is used to run the change script of a window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Run_Window_Value" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Run_Window_Value_Modified

This call is used to run the change script of a modified window field from any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Run_Window_Value_Modified" in dictionary 5261,
Dictionary, "Form", "Window", "Field", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Pull_Window_Focus

This call is used to pull the focus away from a window from any open form in any dictionary. This will force any pending change or post scripts to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Pull_Window_Focus" in dictionary 5261,
Dictionary, "Form", "Window", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Table_Value1

This call is used to obtain the value of a field located in any table in any dictionary using an index containing one field.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1;
l_key1 = "Value1";
call with name "MBS_Get_Table_Value1" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index},
                    "Key1", l_key1;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Set_Table_Value1

This call is used to update the value of a field located in any table in any dictionary using an index containing one field. You can specify whether the creation of a new table record is allowed.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in boolean IN_Allow_Add;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1;
l_key1 = "Value1";
l_field = "Value"
call with name "MBS_Set_Table_Value1" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index}, true
{allow add},
                    "Key1", l_key1;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Table_Value2

This call is used to obtain the value of a field located in any table in any dictionary using an index containing two fields.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2;
l_key1 = "Value1";
l_key2 = "Value2";
call with name "MBS_Get_Table_Value2" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index},
                    "Key1", l_key1,
                    "Key2", l_key2;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Set_Table_Value2

This call is used to update the value of a field located in any table in any dictionary using an index containing two fields. You can specify whether the creation of a new table record is allowed.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in boolean IN_Allow_Add;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2;
l_key1 = "Value1";
l_key2 = "Value2";
l_field = "Value"
call with name "MBS_Set_Table_Value2" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index}, true
{allow add},
                    "Key1", l_key1,
                    "Key2", l_key2;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Table_Value3

This call is used to obtain the value of a field located in any table in any dictionary using an index containing three fields.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
in string IN_Key_Name3;
in anonymous field IN_Key_Value3;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2, l_key3;
l_key1 = "Value1";
l_key2 = "Value2";
l_key3 = "Value3";
call with name "MBS_Get_Table_Value3" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index},
                    "Key1", l_key1,
                    "Key2", l_key2,
                    "Key3", l_key3;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Set_Table_Value3

This call is used to update the value of a field located in any table in any dictionary using an index containing three fields. You can specify whether the creation of a new table record is allowed.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in boolean IN_Allow_Add;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
in string IN_Key_Name3;
in anonymous field IN_Key_Value3;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2, l_key3;
l_key1 = "Value1";
l_key2 = "Value2";
l_key3 = "Value3";
l_field = "Value"
call with name "MBS_Set_Table_Value3" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index}, true
{allow add},
                    "Key1", l_key1,
                    "Key2", l_key2,
                    "Key3", l_key3;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Table_Value4

This call is used to obtain the value of a field located in any table in any dictionary using an index containing four fields.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
out anonymous field OUT_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
in string IN_Key_Name3;
in anonymous field IN_Key_Value3;
in string IN_Key_Name4;
in anonymous field IN_Key_Value4;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2, l_key3, l_key4;
l_key1 = "Value1";
l_key2 = "Value2";
l_key3 = "Value3";
l_key4 = "Value4";
call with name "MBS_Get_Table_Value4" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index},
                    "Key1", l_key1,
                    "Key2", l_key2,
                    "Key3", l_key3,
                    "Key4", l_key4;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Set_Table_Value4

This call is used to update the value of a field located in any table in any dictionary using an index containing four fields. You can specify whether the creation of a new table record is allowed.

All table and field names need to be the technical names and surrounded by single quotes if they contain a space. The status returned will contain the number of errors that occurred, a value of OKAY (zero) means the call was successful. The Key Name fields need to contain the technical names of the segment fields of the index being used.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Table_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
out integer OUT_Status;
in integer IN_Index;
in boolean IN_Allow_Add;
in string IN_Key_Name1;
in anonymous field IN_Key_Value1;
in string IN_Key_Name2;
in anonymous field IN_Key_Value2;
in string IN_Key_Name3;
in anonymous field IN_Key_Value3;
in string IN_Key_Name4;
in anonymous field IN_Key_Value4;
```

An example script is:

```
local integer l_status;
local string l_field;
local string l_key1, l_key2, l_key3, l_key4;
l_key1 = "Value1";
l_key2 = "Value2";
l_key3 = "Value3";
l_key4 = "Value4";
l_field = "Value"
call with name "MBS_Set_Table_Value4" in dictionary 5261,
Dictionary, "Table", "Field", l_field, l_status, 1 {Index}, true
{allow add},
                    "Key1", l_key1,
                    "Key2", l_key2,
                    "Key3", l_key3,
                    "Key4", l_key4;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Table_Buffer_Value

This call is used to obtain the value of a table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout anonymous field INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;

call with name "MBS_Get_Table_Buffer_Value" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_Boolean

This call is used to obtain the value of a boolean table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout boolean INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;

call with name "MBS_Get_Table_Buffer_Value_Boolean" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_Date

This call is used to obtain the value of a date table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout date INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;

call with name "MBS_Get_Table_Buffer_Value_Date" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_Numeric

This call is used to obtain the value of a numeric table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout vcurrency INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;

call with name "MBS_Get_Table_Buffer_Value_Numeric" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_String

This call is used to obtain the value of a string table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout string INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;

call with name "MBS_Get_Table_Buffer_Value_string" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_Text

This call is used to obtain the value of a text table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout text INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;

call with name "MBS_Get_Table_Buffer_Value_Text" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Get_Table_Buffer_Value_Time

This call is used to obtain the value of a time table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
inout time INOUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;

call with name "MBS_Get_Table_Buffer_Value_Time" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status = OKAY then
    warning str(l_field);
end if;
```

## MBS_Set_Table_Buffer_Value

This call is used to update the value of a table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in anonymous field IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;

l_field = "Value";
call with name "MBS_Set_Table_Buffer_Value" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_Boolean

This call is used to update the value of a boolean table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in boolean IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;

l_field = true;
call with name "MBS_Set_Table_Buffer_Value_Boolean" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_Date

This call is used to update the value of a date table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in date IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;

l_field = mkdate(1, 1, 1980);
call with name "MBS_Set_Table_Buffer_Value_Date" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_Numeric

This call is used to update the value of a numeric table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in vcurrency IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;

l_field = 0.00;
call with name "MBS_Set_Table_Buffer_Value_Numeric" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_String

This call is used to update the value of a string table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in string IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;

l_field = "Value";
call with name "MBS_Set_Table_Buffer_Value_String" in dictionary
5261, Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_Text

This call is used to update the value of a text table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in text IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;

l_field = "Value";
call with name "MBS_Set_Table_Buffer_Value_Text" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Table_Buffer_Value_Time

This call is used to update the value of a time table buffer field from any associated table on any open form in any dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in string IN_Field_Name;
in time IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;

l_field = mktime(0, 0 , 0);
call with name "MBS_Set_Table_Buffer_Value_Time" in dictionary 5261,
Dictionary, "Form", "Table", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Copy_To Window

This call is used to copy table buffer fields to the matching window fields on any window on any open form in any dictionary.

Note this function does not check the AutoCopy property of the window fields.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name "MBS_Copy_To_Window" in dictionary 5261, Dictionary,
"Form", "Window", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Copy_From_Window

This call is used to copy table buffer fields from the matching window fields on any window on any open form in any dictionary.

Note this function does not check the AutoCopy property of the window fields.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name "MBS_Copy_From_Window" in dictionary 5261,
Dictionary, "Form", "Window", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Copy_To Window_Modified

This call is used to copy table buffer fields to the matching window fields on any modified window on any open form in any dictionary.

Note this function does not check the AutoCopy property of the window fields.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name "MBS_Copy_To_Window_Modified" in dictionary 5261,
Dictionary, "Form", "Window", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Copy_From_Window_Modified

This call is used to copy table buffer fields from the matching window fields on any modified window on any open form in any dictionary.

Note this function does not check the AutoCopy property of the window fields.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name "MBS_Copy_From_Window_Modified" in dictionary 5261,
Dictionary, "Form", "Window", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Get

This call is used issue a get table or change table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

Operations available (with their values) are:

- GET+FIRST: 11
- GET+PREV: 12
- GET+NEXT: 13
- GET+LAST: 14
- GET+EQUAL: 15
- CHG+FIRST: 21
- CHG+PREV: 22
- CHG+NEXT: 23
- CHG+LAST: 24
- CHG+EQUAL: 25
- CHG+FIRST+LOCK: 51
- CHG+PREV+LOCK: 52
- CHG+NEXT+LOCK: 53
- CHG+LAST+LOCK: 54
- CHG+EQUAL+LOCK: 55

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in integer IN_Operation;
in integer IN_Key;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Get" in dictionary 5261,
Dictionary, "Form", "Table", <Operation>, <Key Number>, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Save

This call is used issue a save table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Save" in dictionary 5261,
Dictionary, "Form", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Remove

This call is used issue a remove table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Remove" in dictionary 5261,
Dictionary, "Form", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Release

This call is used issue a release table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Release" in dictionary 5261,
Dictionary, "Form", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Range

This call is used issue a range table commands against the table buffer of a table associated with any open form in any dictionary.

See full example on the following page.

Operations available (with their values) are:

- RANGE_CLEAR: 1
- RANGE_START: 2
- RANGE_END: 3
- RANGE_REMOVE: 4

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
in integer IN_Operation;
in integer IN_Key;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Range" in dictionary 5261,
Dictionary, "Form", "Table", <Operation>, <Key Number>, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

A more complex example of setting a range and iterating through it is shown in the script below:

```
local integer MBS_Status;
local string MBS_String_Value;

open form RM_Customer_Address;
MBS_String_Value = "ADAMPARK0001";

call with name "MBS_Table_Buffer_Range" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    {RANGE_CLEAR} 1 {Operation},
    1 {Key},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Table_Buffer_Clear" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Set_Table_Buffer_Value_String" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    "'Customer Number'" {Field},
    MBS_String_Value, MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Table_Buffer_Range" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    {RANGE_START} 2 {Operation},
    1 {Key},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Table_Buffer_Fill" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    MBS_Status;
if MBS_Status <> OKAY then
```

```
    debug str(MBS_Status);
end if;

call with name "MBS_Set_Table_Buffer_Value_String" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    "'Customer Number'" {Field},
    MBS_String_Value, MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Table_Buffer_Range" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    {RANGE_END} 3 {Operation},
    1 {Key},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

call with name "MBS_Table_Buffer_Get" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    {CHG+FIRST} 21 {Operation},
    1 {Key},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;

while MBS_Status = OKAY do
    call with name "MBS_Get_Table_Buffer_Value_String" in dictionary 5261,
        0 {Dict},
        "RM_Customer_Address" {Form},
        "RM_Customer_MSTR_ADDR" {Table},
        "'Address Code'" {Field},
        MBS_String_Value, MBS_Status;
    if MBS_Status = OKAY then
{        warning str(MBS_String_Value);
}    end if;


    case ask("Select action to perform on " + MBS_String_Value, "Save",
"Remove", "Release")
        in [ASKBUTTON1]
            call with name "MBS_Table_Buffer_Save" in dictionary 5261,
                0 {Dict},
                "RM_Customer_Address" {Form},
                "RM_Customer_MSTR_ADDR" {Table},
                MBS_Status;
            if MBS_Status <> OKAY then
```

```
                debug str(MBS_Status);
            end if;
        in [ASKBUTTON2]
            call with name "MBS_Table_Buffer_Remove" in dictionary 5261,
                0 {Dict},
                "RM_Customer_Address" {Form},
                "RM_Customer_MSTR_ADDR" {Table},
                MBS_Status;
            if MBS_Status <> OKAY then
                debug str(MBS_Status);
            end if;
        else
            call with name "MBS_Table_Buffer_Release" in dictionary 5261,
                0 {Dict},
                "RM_Customer_Address" {Form},
                "RM_Customer_MSTR_ADDR" {Table},
                MBS_Status;
            if MBS_Status <> OKAY then
                debug str(MBS_Status);
            end if;
    end case;

    call with name "MBS_Table_Buffer_Get" in dictionary 5261,
        0 {Dict},
        "RM_Customer_Address" {Form},
        "RM_Customer_MSTR_ADDR" {Table},
        {CHG+NEXT} 23 {Operation},
        1 {Key},
        MBS_Status;
    if MBS_Status <> OKAY then
        debug str(MBS_Status);
    end if;

end while;

call with name "MBS_Table_Buffer_Clear" in dictionary 5261,
    0 {Dict},
    "RM_Customer_Address" {Form},
    "RM_Customer_MSTR_ADDR" {Table},
    MBS_Status;
if MBS_Status <> OKAY then
    debug str(MBS_Status);
end if;
out boolean OUT_Condition;
```

## MBS_Table_Buffer_Clear

This call is used issue a clear table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Clear" in dictionary 5261,
Dictionary, "Form", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Table_Buffer_Fill

This call is used issue a fill table command against the table buffer of a table associated with any open form in any dictionary.

See full example in the MBS_Table_Buffer_Range helper function section.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Table_Name;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;

call with name " MBS_Table_Buffer_Fill" in dictionary 5261,
Dictionary, "Form", "Table", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Runtime_Execute

This call is used to execute Dexterity sanScript in the context of the specified dictionary.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute" in dictionary 5261, l_text,
l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Background

This call is used to execute Dexterity sanScript in the context of the specified dictionary after any background processes by adding it to the background queue.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Background" in dictionary 5261,
l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Delayed

This call is used to execute Dexterity sanScript in the context of the specified dictionary after all foreground scripts have completed by running it delayed.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Delayed" in dictionary 5261,
l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_After_Background

This call is used to execute Dexterity sanScript in the context of the specified dictionary in the foreground, but after any background processes by adding it to the background queue.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_After_Background" in dictionary
5261, l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Modified

This call is used to execute Dexterity sanScript in the context of the specified modified dictionary.

This allows Dexterity to reference Modifier added local fields.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Modified" in dictionary 5261,
l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Modified_Background

This call is used to execute Dexterity sanScript in the context of the specified modified dictionary after any background processes by adding it to the background queue.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Modified_Background" in
dictionary 5261, l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Modified_Delayed

This call is used to execute Dexterity sanScript in the context of the specified modified dictionary after all foreground scripts have completed by running it delayed.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Modified_Delayed" in dictionary
5261, l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_Runtime_Execute_Modified_After_Background

This call is used to execute Dexterity sanScript in the context of the specified modified dictionary in the foreground, but after any background processes by adding it to the background queue.

The parameter list for this call is:

```
inout text INOUT_Text;
in integer IN_Prod_ID;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_dict;

clear l_text;
l_text = l_text + "warning ""Hello World"";" + char(13);
l_dict = 0; {Dictionary}
call with name "MBS_Runtime_Execute_Modified_After_Background" in
dictionary 5261, l_text, l_dict, l_status;
if l_status <> OKAY then
    warning l_text;
end if;
```

## MBS_SQL_Set_Database

This call is used to change the default database from the current Company database to the specified database. It will only affect the next SQL script to be executed using Helper Functions.

The parameter list for this call is:

```
in string IN_DB;
```

An example script is:

```
call with name "MBS_SQL_Set_Database" in dictionary 5261,
"<DBNAME>";
```

## MBS_SQL_Check_Exists

This call is used to execute a SQL Select statement in the context of the current company database and indicate whether any data records were returned.

This helper function has been replaced by the MBS_SQL_Execute Helper Function.

The text field returned will contain the error message, or the number of records returned with or without data depending on the options passed in.

Use the MBS_SQL_Parse_Data series of Helper Functions to easily parse data back into fields and convert datatypes as required.

The parameter list for this call is:

```
inout text INOUT_TSQL;
in boolean IN_Return_Data;
in boolean IN_Return_Columns;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Check_Exists" in dictionary 5261, l_text,
true, true, l_status;
case l_status
    in [OKAY] {Data}
        warning l_text;
    in [MISSING] {No Data}
        warning l_text;
    in [EOF] {Overflow}
        warning "Overflow" + char(13) + l_text;
    else {Error}
        warning l_text;
end case;
```

A more complex example of running a query iterating through the resulting rows and columns is shown in the script below:

```
local text MBS_Text_Field;
local integer MBS_Status;

local string l_line;
local string l_field;
local integer l_pos, l_old_pos;
local long l_row;
local integer l_column;
local string l_ID, l_Name, l_Contact, l_Address;
```

```
call with name "MBS_Script_Load_SQL" in dictionary 5261,
    "CUSTOMERS", MBS_Text_Field;

call with name "MBS_SQL_Check_Exists" in dictionary 5261,
    MBS_Text_Field, true {Return Data}, false {Show Names}, MBS_Status;
case MBS_Status
    in [OKAY] {Data}
{        warning MBS_Text_Field;
}        l_row = 0;
        l_pos = 1;
        repeat
            l_old_pos = l_pos;
            l_pos = pos(MBS_Text_Field, char(13), l_old_pos);
            if l_pos > 0 then
                l_line = substring(MBS_Text_Field, l_old_pos, min(255, l_pos-l_old_pos));
                l_pos = l_pos + 1;
                if not empty(l_line) then
                    increment l_row;
                    clear l_ID, l_Name, l_Contact, l_Address;
                    for l_column = 1 to 4 do
                        l_field = trim(RW_Token(l_line, char(9), l_column));
                        case l_column
                            in [1]
                                l_ID = l_field;
                            in [2]
                                l_Name = l_field;
                            in [3]
                                l_Contact = l_field;
                            in [4]
                                l_Address = l_field;
                            else
                        end case;
                    end for;
                    warning text("Record: " + str(l_row) + char(13) + "ID: " + l_ID + char(13)
                                + "Name: " + l_Name + char(13) + "Contact: " + l_Contact + char(13)
                                + "Address: " + l_Address + char(13));
                end if;
            end if;
        until l_pos = 0 or empty(l_line);

    in [MISSING] {No Data}
        warning MBS_Text_Field;
    in [EOF] {Overflow}
        warning "Overflow" + char(13) + MBS_Text_Field;
    else {Error}
        warning MBS_Text_Field;
end case;
```

Note that conversion of string formatted data back to other data types should now use the MBS_Convert series of Helper Functions rather than use the manually code examples below.

Below are examples showing how to manually convert the string formatted data back to other data types:

```
{ Example Code for other Datatypes }

local integer l_Integer;
l_Integer = integer(value(l_field));

local currency l_Currency;
l_Currency = currency(value(l_field));

local boolean l_Boolean;
l_Boolean = (upper(l_field) = TRUE_STRING); { When stored as TRUE or FALSE }
l_Boolean = (value(l_field) <> 0); { When stored as 0 or 1 }

local integer l_year, l_month, l_day;
local date l_Date;
l_year = integer(value(substring(l_field, 1, 4)));
l_month = integer(value(substring(l_field, 6, 2)));
l_day = integer(value(substring(l_field, 9, 2)));
l_Date = mkdate(l_month, l_day, l_year); { Where Date in 'YYYY/MM/DD' format }

local integer l_hour, l_minute, l_second, l_pos;
local time l_Time;
l_pos = pos(l_field, CH_COLON, 1); { Find colon in Datetime string }
l_hour = integer(value(substring(l_field, l_pos-2, 2)));
l_minute = integer(value(substring(l_field, l_pos+1, 2)));
l_second = integer(value(substring(l_field, l_pos+4, 2)));
l_Time = mktime(l_hour, l_minute, l_second); { Where Time in 'HH:MM:SS' format }
```

## MBS_SQL_Execute

This call is used to execute a SQL Select statement in the context of the selected database and indicate whether any data records were returned.

This helper function replaces the MBS_SQL_Check_Exists Helper Function.

The text field returned will contain the error message, or the number of records returned with or without data depending on the options passed in.

Use the MBS_SQL_Parse_Data series of Helper Functions to easily parse data back into fields and convert datatypes as required.

The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_DB;
in boolean IN_Return_Data;
in boolean IN_Return_Columns;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local string l_db;
l_db = 'Intercompany ID' of globals;
clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Execute" in dictionary 5261, l_text, l_db,
true, true, l_status;
case l_status
    in [OKAY] {Data}
        warning l_text;
    in [MISSING] {No Data}
        warning l_text;
    in [EOF] {Overflow}
        warning "Overflow" + char(13) + l_text;
    else {Error}
        warning l_text;
end case;
```

An example of running a query iterating through the resulting rows and columns is shown in the script below:

```
local text MBS_Text_Field;
local integer MBS_Status;
local string MBS_Database;

local integer l_row;
local string l_ID, l_Name, l_Contact, l_Address;
local string MBS_SQL_String_Value;

call with name "MBS_Script_Load_SQL_DB" in dictionary 5261, "CUSTOMERS",
    MBS_Text_Field, MBS_Database;

call with name "MBS_SQL_Execute" in dictionary 5261,
    MBS_Text_Field, MBS_Database, true {Return Data}, false {Show Names}, MBS_Status;
case MBS_Status
    in [OKAY] {Data}
{        warning MBS_Text_Field;
}      l_row = 0;
       repeat
          call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,.
             l_row+1 {Row}, 1 {Col}, MBS_SQL_String_Value, MBS_Status;
          if MBS_Status = OKAY then
             increment l_row;
             l_ID = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 2 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Name = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 3 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Contact = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 4 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Address = MBS_SQL_String_Value;
             warning text("Record: " + str(l_row) + char(13) + "ID: " + l_ID + char(13)
                      + "Name: " + l_Name + char(13) + "Contact: " + l_Contact + char(13)
                      + "Address: " + l_Address + char(13));
          end if;
       until MBS_Status <> OKAY;

    in [MISSING] {No Data}
       warning MBS_Text_Field;
    in [EOF] {Overflow}
       warning "Overflow" + char(13) + MBS_Text_Field;
    else {Error}
       warning MBS_Text_Field;
end case;
```

## MBS_SQL_Get_Data

This call is used to execute a SQL Select statement in the context of the selected database and indicate whether any data records were returned.

This helper function uses SQL Server to format the data into a tab delimited data set. It can be up to 10 times faster than the MBS_SQL_Check_Exists or MBS_SQL_Execute Helper Functions.

The text field returned will only contain the data returned without any header row.

Use the MBS_SQL_Parse_Data series of Helper Functions to easily parse data back into fields and convert datatypes as required.

The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_DB;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local string l_db;
l_db = 'Intercompany ID' of globals;
clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Get_Data" in dictionary 5261, l_text, l_db,
l_status;
case l_status
    in [OKAY] {Data}
        warning l_text;
    in [MISSING] {No Data}
        warning l_text;
    in [EOF] {Overflow}
        warning "Overflow" + char(13) + l_text;
    else {Error}
        warning l_text;
end case;
```

An example of running a query iterating through the resulting rows and columns is shown in the script below:

```
local text MBS_Text_Field;
local integer MBS_Status;
local string MBS_Database;

local integer l_row;
local string l_ID, l_Name, l_Contact, l_Address;
local string MBS_SQL_String_Value;

call with name "MBS_Script_Load_SQL_DB" in dictionary 5261, "CUSTOMERS",
    MBS_Text_Field, MBS_Database;

call with name "MBS_SQL_Get_Data" in dictionary 5261, MBS_Text_Field, MBS_Database, MBS_Status;
case MBS_Status
    in [OKAY] {Data}
{        warning MBS_Text_Field;
}      l_row = 0;
       repeat
          call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,.
             l_row+1 {Row}, 1 {Col}, MBS_SQL_String_Value, MBS_Status;
          if MBS_Status = OKAY then
             increment l_row;
             l_ID = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 2 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Name = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 3 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Contact = MBS_SQL_String_Value;
             call with name "MBS_SQL_Parse_Data_String" in dictionary 5261, MBS_Text_Field,
                 l_row {Row}, 4 {Col}, MBS_SQL_String_Value, MBS_Status;
             l_Address = MBS_SQL_String_Value;
             warning text("Record: " + str(l_row) + char(13) + "ID: " + l_ID + char(13)
                      + "Name: " + l_Name + char(13) + "Contact: " + l_Contact + char(13)
                      + "Address: " + l_Address + char(13));
          end if;
       until MBS_Status <> OKAY;

   in [MISSING] {No Data}
      warning MBS_Text_Field;
   in [EOF] {Overflow}
      warning "Overflow" + char(13) + MBS_Text_Field;
   else {Error}
      warning MBS_Text_Field;
end case;
```

## MBS_SQL_Parse_Data

This call is used to parse a field of any datatype from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out anonymous field OUT_Data;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local string l_data;

call with name "MBS_SQL_Parse_Data" in dictionary 5261, l_text, 1
{Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Boolean

This call is used to parse a Boolean field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out boolean OUT_Boolean;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local boolean l_data;

call with name "MBS_SQL_Parse_Data_Boolean" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Currency

This call is used to parse a Currency field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out currency OUT_Currency;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local currency l_data;

call with name "MBS_SQL_Parse_Data_Currency" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Date

This call is used to parse a Date field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out date OUT_Date;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local date l_data;

call with name "MBS_SQL_Parse_Data_Date" in dictionary 5261, l_text,
1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Datetime

This call is used to parse a Datetime field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out datetime OUT_Datetime;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local datetime l_data;

call with name "MBS_SQL_Parse_Data_Datetime" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Integer

This call is used to parse an Integer field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out integer OUT_Integer;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local integer l_data;

call with name "MBS_SQL_Parse_Data_Integer" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Long

This call is used to parse a Long field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out long OUT_Long;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local long l_data;

call with name "MBS_SQL_Parse_Data_Long" in dictionary 5261, l_text,
1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_String

This call is used to parse a String field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out string OUT_String;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local string l_data;

call with name "MBS_SQL_Parse_Data_String" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Text

This call is used to parse a Text field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out text OUT_Text;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local text l_data;

call with name "MBS_SQL_Parse_Data_Text" in dictionary 5261, l_text,
1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Time

This call is used to parse a Time field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out time OUT_Time;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local time l_data;

call with name "MBS_SQL_Parse_Data_Time" in dictionary 5261, l_text,
1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_VCurrency

This call is used to parse a VCurrency field from a tab delimited data field.

It can be used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Row;
in integer IN_Col;
out vcurrency OUT_VCurrency;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_text;
local vcurrency l_data;

call with name "MBS_SQL_Parse_Data_VCurrency" in dictionary 5261,
l_text, 1 {Row}, 1 {Col}, l_data, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_SQL_Parse_Data_Reset

This call is used to reset the counters used when parsing tab delimited data in a text field, so that the parsing starts at the start of the text field.

It is called automatically and so is not required when used with the MBS_SQL_Check_Exists, MBS_SQL_Execute, MBS_SQL_Get_Data and MBS_Copy_From_Clipboard Helper Functions.

The parameter list for this call is:

```
<None>;
```

An example script is:

```
call with name "MBS_SQL_Parse_Data_Reset" in dictionary 5261;
```

## MBS_Export_SQL_Query_To_File

This call is used to execute a SQL Select statement in the context of the current company database and export the result set as a text file.

The parameter list for this call is:

```
inout text INOUT_Code;
inout string INOUT_Pathname;
in boolean IN_Header;
in boolean IN_Quotes;
in integer IN_Mode; { 0 - CSV, 1 - Tab, 2 - User Defined }
in string IN_Delimiter;
in boolean IN_Append;
out long OUT_Rows;
out integer OUT_Status;
```

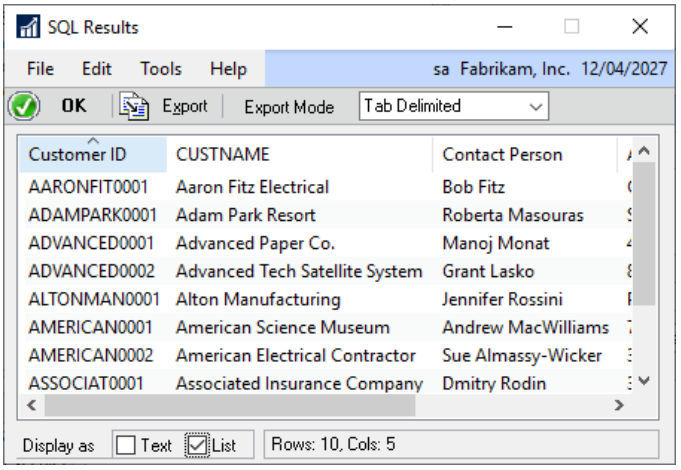
An example script is:

```
local integer l_status;
local text l_text;
local string l_path;
local long l_rows;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_Export_SQL_Query_To_File" in dictionary 5261,
l_text, l_path, true {Header}, true {Quotes}, 0 {CSV}, ""
{Delimiter}, false, {Append}, l_rows, l_status;
if l_status = OKAY then
    l_text = str(l_rows) + " rows exported to " + l_path + ".";
    warning l_text;
end if;
```

## MBS_SQL_Results

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a SQL Results window. The results can be exported from this window if desired.




The parameter list for this call is:

```
inout text INOUT_TSQL;
```

An example script is:

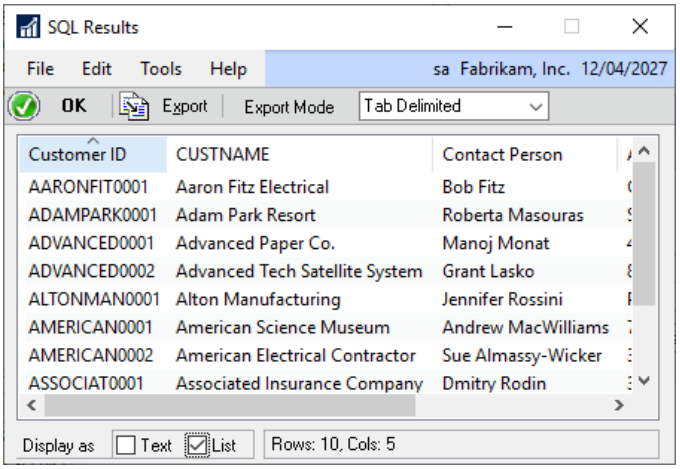
```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results" in dictionary 5261, l_text;
```

## MBS_SQL_Results_Immediate

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a SQL Results window. The results can be exported from this window if desired.

The Immediate version of this call displays the result set immediately instead of delayed and so is useful if you want to display data within a script and ask for the next action using a system dialog.




The parameter list for this call is:

```
inout text INOUT_TSQL;
```

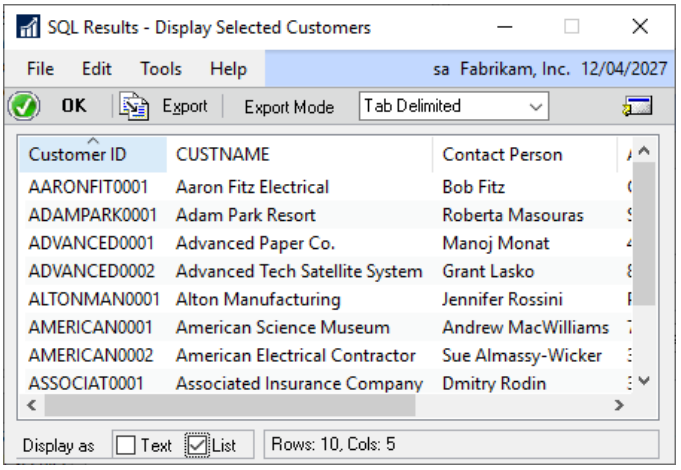
An example script is:

```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Immediate" in dictionary 5261,
l_text;
```

## MBS_SQL_Results_Goto

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a SQL Results window. The results can be exported from this window if desired or further actions can be started using the SQL Gotos.




The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_ScriptID;
```

An example script is:

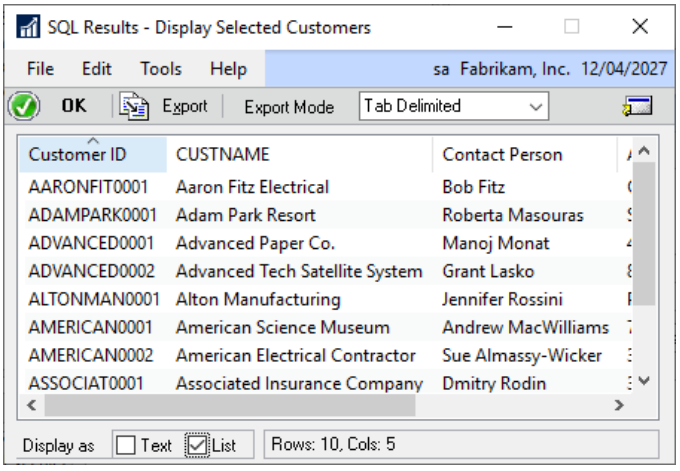
```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Goto" in dictionary 5261, l_text,
"ScriptID";
```

## MBS_SQL_Results_Immediate_Goto

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a SQL Results window. The results can be exported from this window if desired or further actions can be started using the SQL Gotos.

The Immediate version of this call displays the result set immediately instead of delayed and so is useful if you want to display data within a script and ask for the next action using a system dialog.




The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_ScriptID;
```

An example script is:

```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Immediate_Goto" in dictionary 5261,
l_text, "ScriptID";
```

## MBS_SQL_Results_Close

This call is to close the SQL Results window.

The parameter list for this call is:

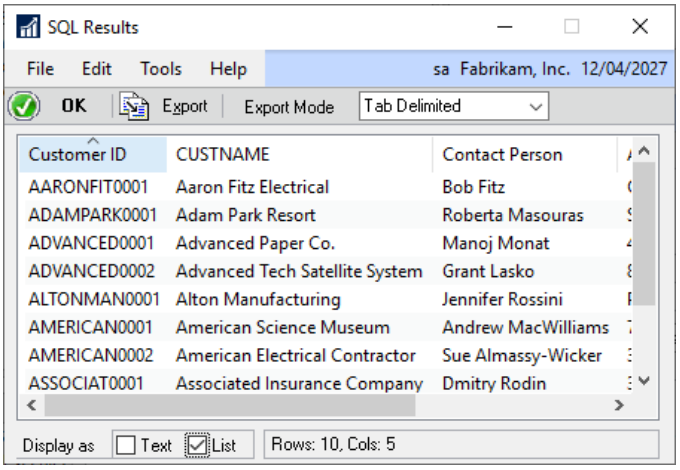
```
<None>
```

An example script is:

```
call with name "MBS_SQL_Results_Close" in dictionary 5261;
```

## MBS_SQL_Results2

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a second SQL Results 2 window. The results can be exported from this window if desired.




The parameter list for this call is:

```
inout text INOUT_TSQL;
```

An example script is:

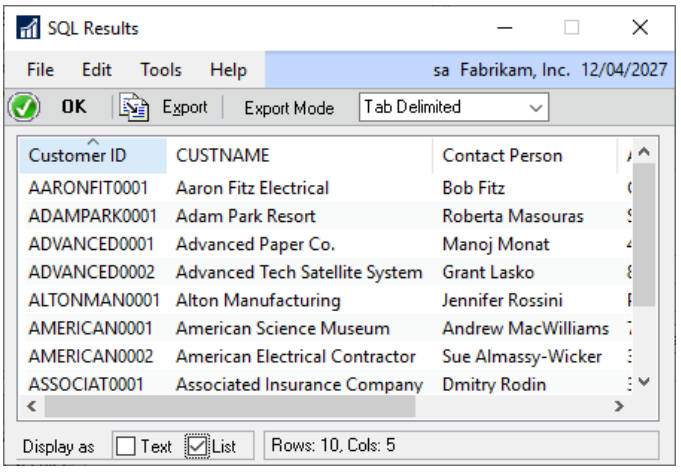
```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results2" in dictionary 5261, l_text;
```

## MBS_SQL_Results_Immediate2

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a second SQL Results 2 window. The results can be exported from this window if desired.

The Immediate version of this call displays the result set immediately instead of delayed and so is useful if you want to display data within a script and ask for the next action using a system dialog.




The parameter list for this call is:

```
inout text INOUT_TSQL;
```

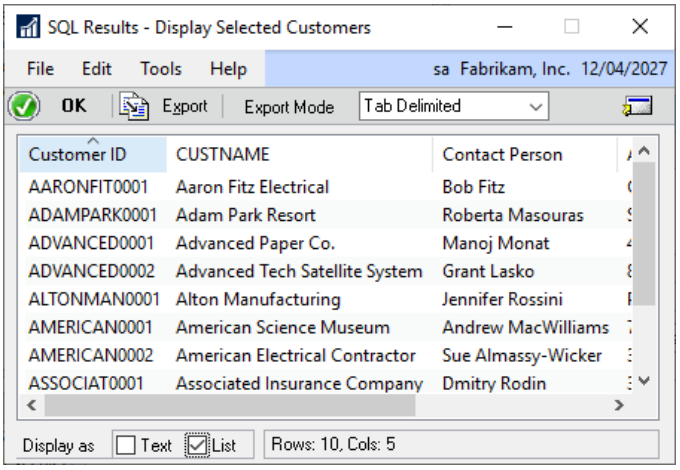
An example script is:

```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Immediate2" in dictionary 5261,
l_text;
```

## MBS_SQL_Results_Goto2

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a second SQL Results 2 window. The results can be exported from this window if desired or further actions can be started using the SQL Gotos.




The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_ScriptID;
```

An example script is:

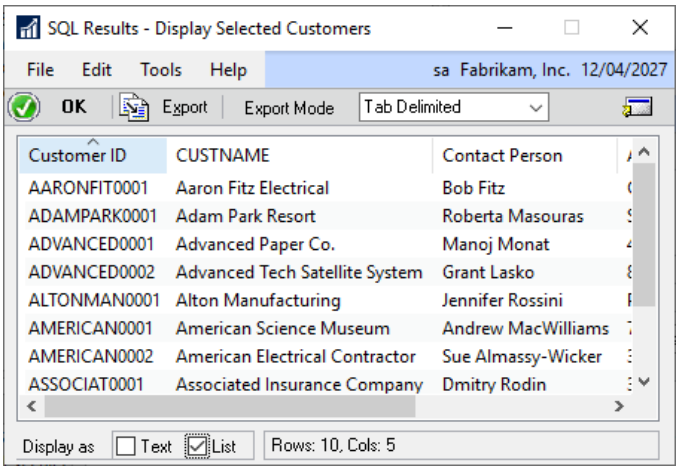
```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Goto2" in dictionary 5261, l_text,
"ScriptID";
```

## MBS_SQL_Results_Immediate_Goto2

This call is used to execute a SQL Select statement in the context of the current company database and display the results returned in a second SQL Results 2 window. The results can be exported from this window if desired or further actions can be started using the SQL Gotos.

The Immediate version of this call displays the result set immediately instead of delayed and so is useful if you want to display data within a script and ask for the next action using a system dialog.




The parameter list for this call is:

```
inout text INOUT_TSQL;
in string IN_ScriptID;
```

An example script is:

```
local text l_text;

clear l_text;
l_text = l_text + "select * from table" + char(13);
call with name "MBS_SQL_Results_Immediate_Goto2" in dictionary 5261,
l_text, "ScriptID";
```

## MBS_SQL_Results_Close2

This call is to close the second SQL Results 2 window.

The parameter list for this call is:

```
<None>
```

An example script is:

```
call with name "MBS_SQL_Results_Close2" in dictionary 5261;
```

## MBS_SQL_Goto_Get_Data

This call is used in a Runtime Execute Setup script to retrieve data from a SQL result set for use with SQL Gotos.

The parameter list for this call is:

```
in integer IN_Window;
in long IN_Position;
in string IN_Column;
in integer IN_Type; { 1 = string, 2 = long, 3 = currency, 4 = date, 5 = time }
out anonymous field OUT_Field;
```

An example script is:

```
in integer MBS_SQLGotoWindow;
in long MBS_SQLGotoCount;
local long MBS_SQLGotoPos;
local string MBS_SQLGotoValue1;
local string MBS_Message;

if MBS_SQLGotoCount > 0 then
    for MBS_SQLGotoPos = 1 to MBS_SQLGotoCount do
        call with name "MBS_SQL_Goto_Get_Data" in dictionary 5261,
            MBS_SQLGotoWindow, MBS_SQLGotoPos, "Customer ID" { Column Label },
            1 { Column Datatype: 1 = string, 2 = long, 3 = currency, 4 = date, 5 = time },
            MBS_SQLGotoValue1;
    end for;
end if;
```

## MBS_SQL_Goto_Close

This call is used in a Runtime Execute Setup script to close or clear the SQL result set for use with SQL Gotos.

The parameter list for this call is:

```
in integer IN_Window;
```

An example script is:

```
in integer MBS_SQLGotoWindow;

call with name "MBS_SQL_Goto_Close" in dictionary 5261, MBS_SQLGotoWindow;
```

## MBS_SQL_Sort_Get

This call is used in a Runtime Execute Setup script to read the current sort column for a result set for use with SQL Gotos.

This Helper Function can be called from a SQL Goto script running with the Goto Mode. of Before Close to get the current sort column and order values so they can saved to be used later.

The parameter list for this call is:

```
in integer IN_Window;
out string OUT_Column;
out integer OUT_Mode; {0 = Ascending, 1 = Descending}
```

An example script is:

```
in integer MBS_SQLGotoWindow;
local string l_column;
local integer l_mode;

call with name "MBS_SQL_Sort_Get" in dictionary 5261, MBS_SQLGotoWindow, l_column, l_mode;
```

## MBS_SQL_Sort_Set

This call is used in a Runtime Execute Setup script to change the current sort column for a result set for use with SQL Gotos.

This Helper Function can be called from a SQL Goto script running with the Goto Mode. of After Query to set the current sort column and order based on previously saved values.

The parameter list for this call is:

```
in integer IN_Window;
in string OUT_Column;
in integer OUT_Mode; {0 = Ascending, 1 = Descending}
```

An example script is:

```
in integer MBS_SQLGotoWindow;

call with name "MBS_SQL_Sort_Set" in dictionary 5261, MBS_SQLGotoWindow, "Customer ID", 0;
```

## MBS_SQL_Export_Data

This call is used in a Runtime Execute Setup script to export the SQL result set to a text field for use with SQL Gotos.

If exported as tab delimited using the TABFILE mode, you can parse the resulting data with the SQL Parsing helper functions.

If exported as HTML using the HTMLFILE mode, you can use result set in an email using the MBS_Email_API as long as the email is in HTML mode. Use TABFILE mode if not sending emails using HTML.

The parameter list for this call is:

```
in integer IN_Window;
in integer IN_FileMode; {TABFILE = 6, COMMAFILE = 7, HTMLFILE = 12}
out text OUT_Text;
out integer OUT_Status;
in boolean IN_Selected; {true = Selected only, false = all records}
```

An example script is:

```
in integer MBS_SQLGotoWindow;
local text l_text;

call with name "MBS_SQL_Export_Data" in dictionary 5261, MBS_SQLGotoWindow, TABFILE,
l_text, l_status, true;
```

## MBS_Net_Execute

This call is used to execute .Net scripts (Visual C# or Visual Basic.Net).

The parameter list for this call is:

```
in integer IN_Mode;
inout text INOUT_References;
inout text INOUT_Script;
inout text INOUT_Results;
out boolean OUT_Success;
```

An example Visual C# script is:

```
local integer MBS_Mode;
local text MBS_References;
local text MBS_Script;
local text MBS_Results;
local boolean MBS_Success;

MBS_Success = false;
MBS_Mode = 1; {1 for C#, 2 for VB}

clear MBS_Script;
MBS_Script = MBS_Script + "using System.Windows.Forms;" + char(13);
MBS_Script = MBS_Script + "using Microsoft.Dexterity.Bridge;" + char(13);
MBS_Script = MBS_Script + "using Microsoft.Dexterity.Applications;" + char(13);
MBS_Script = MBS_Script + "using Microsoft.Dexterity.Applications.DynamicsDictionary;" + char(13);
MBS_Script = MBS_Script + "using Microsoft.Dexterity.Applications.GpPowerToolsDictionary;" + char(13);

MBS_Script = MBS_Script + "namespace NetExecute" + char(13);
MBS_Script = MBS_Script + "{" + char(13);
MBS_Script = MBS_Script + "    public class Program" + char(13);
MBS_Script = MBS_Script + "    {" + char(13);
MBS_Script = MBS_Script + "        public void Run()" + char(13);
MBS_Script = MBS_Script + "        {" + char(13);
MBS_Script = MBS_Script + "            MessageBox.Show(""Hello from C#"");" + char(13);
MBS_Script = MBS_Script + "        }" + char(13);
MBS_Script = MBS_Script + "    }" + char(13);
MBS_Script = MBS_Script + "}" + char(13);

clear MBS_References;
MBS_References = MBS_References + "System.Windows.Forms.dll" + char(13);
MBS_References = MBS_References + "..\Application.Dynamics.dll" + char(13);
MBS_References = MBS_References + "..\Microsoft.Dexterity.Bridge.dll" + char(13);
MBS_References = MBS_References + "..\Microsoft.Dexterity.Shell.dll" + char(13);
MBS_References = MBS_References + "..\Application.GpPowerTools.dll" + char(13);

call with name "MBS_Net_Execute" in dictionary 5261,
    MBS_Mode, MBS_References, MBS_Script, MBS_Results, MBS_Success;
if not MBS_Success then
    warning MBS_Results;
end if;
```

An example Visual Basic.Net script is:

```
local integer MBS_Mode;
local text MBS_References;
local text MBS_Script;
local text MBS_Results;
local boolean MBS_Success;

MBS_Success = false;
MBS_Mode = 2; {1 for C#, 2 for VB}

clear MBS_Script;
MBS_Script = MBS_Script + "Imports System.Windows.Forms" + char(13);
MBS_Script = MBS_Script + "Imports Microsoft.VisualBasic" + char(13);
MBS_Script = MBS_Script + "Imports Microsoft.Dexterity.Bridge" + char(13);
MBS_Script = MBS_Script + "Imports Microsoft.Dexterity.Applications" + char(13);
MBS_Script = MBS_Script + "Imports Microsoft.Dexterity.Applications.DynamicsDictionary" + char(13);
MBS_Script = MBS_Script + "Imports Microsoft.Dexterity.Applications.GpPowerToolsDictionary" + char(13);

MBS_Script = MBS_Script + "Namespace NetExecute" + char(13);
MBS_Script = MBS_Script + "    Class Program" + char(13);
MBS_Script = MBS_Script + "        Public Function Run() As Object" + char(13);
MBS_Script = MBS_Script + "            MessageBox.Show(""Hello from VB.Net"")" + char(13);
MBS_Script = MBS_Script + "        End Function" + char(13);
MBS_Script = MBS_Script + "    End Class" + char(13);
MBS_Script = MBS_Script + "End Namespace" + char(13);

clear MBS_References;
MBS_References = MBS_References + "System.Windows.Forms.dll" + char(13);
MBS_References = MBS_References + "..\Application.Dynamics.dll" + char(13);
MBS_References = MBS_References + "..\Microsoft.Dexterity.Bridge.dll" + char(13);
MBS_References = MBS_References + "..\Microsoft.Dexterity.Shell.dll" + char(13);
MBS_References = MBS_References + "..\Application.GpPowerTools.dll" + char(13);

call with name "MBS_Net_Execute" in dictionary 5261,
    MBS_Mode, MBS_References, MBS_Script, MBS_Results, MBS_Success;
if not MBS_Success then
    warning MBS_Results;
end if;
```

## MBS_Script_Load_Dex

This call is used to load a Dexterity sanScript script from a Runtime Execute Setup Script ID. It is designed to be used with the MBS_Runtime_Execute Helper Function.

The parameter list for this call is:

```
in string IN_ScriptID;
inout text INOUT_Text;
inout integer INOUT_Dict;
```

An example script is:

```
local text l_text;
local integer l_dict;

call with name "MBS_Script_Load_Dex" in dictionary 5261, "XXXX",
l_text, l_dict;
```

## MBS_Script_Load_SQL

This call is used to load a SQL script from a SQL Execute Setup Script ID. It is designed to be used with the MBS_SQL_Check_Exists Helper Function.

This helper function has been replaced by the MBS_Script_Load_SQL_DB Helper Function.

The parameter list for this call is:

```
in string IN_ScriptID;
inout text INOUT_Text;
```

An example script is:

```
local text l_text;

call with name "MBS_Script_Load_SQL" in dictionary 5261, "XXXX",
l_text;
```

## MBS_Script_Load_SQL_DB

This call is used to load a SQL script and Database context from a SQL Execute Setup Script ID. It is designed to be used with the MBS_SQL_Execute Helper Function.

This helper function replaces the MBS_Script_Load_SQL Helper Function.

The parameter list for this call is:

```
in string IN_ScriptID;
inout text INOUT_Text;
out string OUT_DB;;
```

An example script is:

```
local text l_text;
local string l_db;

call with name "MBS_Script_Load_SQL_DB" in dictionary 5261, "XXXX",
l_text, l_db;
```

## MBS_Script_Load_Net

This call is used to load a Visual C# or Visual Basic.Net script from a .Net Execute Setup Script ID. It is designed to be used with the MBS_Net_Execute Helper Function.

The parameter list for this call is:

```
in string IN_ScriptID;
out integer IN_Mode;
inout text INOUT_References;
inout text INOUT_Script;
```

An example script is:

```
local integer MBS_Mode;
local text MBS_References;
local text MBS_Script;


call with name "MBS_Script_Load_Net" in dictionary 5261,
    "XXXX", MBS_Mode, MBS_References, MBS_Script;
```

## MBS_Param_Set

This call is used to store a value in the DUOS SY_User_Object_Store (SY90000) table which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Param_Get Helper Functions as a method of passing parameters.

The parameter list for this call is:

```
in string IN_Parameter;
in string IN_Value;
```

An example script is:

```
local string l_string;

l_string = "Value";
call with name "MBS_Param_Set" in dictionary 5261, "Variable",
l_string;
```

## MBS_Param_Get

This call is used to read a previously set value from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_Runtime_Execute and MBS_Param_Set Helper Functions as a method of passing parameters.

The parameter list for this call is:

```
in string IN_Parameter;
out string OUT_Value;
```

An example script is:

```
local string l_string;

call with name "MBS_Param_Get" in dictionary 5261, "Variable",
l_string;
```

## MBS_Param_Del

This call is used to remove a previously set value from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_Runtime_Execute and MBS_Param_Set Helper Functions as a method of passing parameters.

The parameter list for this call is:

```
in string IN_Parameter;
```

An example script is:

```
call with name "MBS_Param_Del" in dictionary 5261, "Variable";
```

## MBS_Param_DelAll

This call is used to remove all previously stored parameter values for the current user from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_Runtime_Execute and MBS_Param_Set Helper Functions.

There is no parameter list for this call.

An example script is:

```
call with name "MBS_Param_DelAll" in dictionary 5261;
```

## MBS_Memory_Set

This call is used to store a value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

String data can be up to 132 characters per variable.

The parameter list for this call is:

```
in string IN_Parameter;
in anonymous field IN_Value;
```

An example script is:

```
local string l_value;

l_value = "Value";
call with name "MBS_Memory_Set" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Set_Boolean

This call is used to store a boolean value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Boolean Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
in boolean IN_Value;
```

An example script is:

```
local boolean l_value;

l_value = true
call with name "MBS_Memory_Set_Boolean" in dictionary 5261,
"Variable", l_value;
```

## MBS_Memory_Set_Currency

This call is used to store a currency value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Currency Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
in currency IN_Value;
```

An example script is:

```
local currency l_value;

l_value = 0.00;
call with name "MBS_Memory_Set_Currency" in dictionary 5261,
"Variable", l_value;
```

## MBS_Memory_Set_Date

This call is used to store a date value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Date Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
in date IN_Value;
```

An example script is:

```
local date l_value;

l_value = mkdate(1, 1, 1980);
call with name "MBS_Memory_Set_Date" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Set_Long

This call is used to store an integer or long value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Long Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
in long IN_Value;
```

An example script is:

```
local long l_value;

l_value = 0;
call with name "MBS_Memory_Set_Long" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Set_String

This call is used to store a string value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_String Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

String data can be up to 132 characters per variable.

The parameter list for this call is:

```
in string IN_Parameter;
in string IN_Value;
```

An example script is:

```
local string l_value;

l_value = "Value";
call with name "MBS_Memory_Set_String" in dictionary 5261,
"Variable", l_value;
```

## MBS_Memory_Set_Time

This call is used to store a time value in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Time Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
in time IN_Value;
```

An example script is:

```
local time l_value;

l_value = mktime(0, 0, 0);
call with name "MBS_Memory_Set_Time" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Set_Reference

This call is used to store a table reference in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Reference Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
in reference IN_Reference;
```

An example script is:

```
local reference l_reference;

assign l_reference as reference to table XXXX;
call with name "MBS_Memory_Set_Reference" in dictionary 5261,
"Variable", l_reference;
```

## MBS_Memory_Set_Table

This call is used to store a table reference in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Reference Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
inout anonymous table INOUT_Table;
```

An example script is:

```
local anonymous table l_table;

open table l_table with name "RM_Customer_MSTR";
call with name "MBS_Memory_Set_Table" in dictionary 5261,
"Variable", l_table;
```

## MBS_Memory_Set_Field

This call is used to store a field reference in a memory variable slot which can then be read by another script. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Get_Reference Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
inout anonymous field INOUT_Field;
```

An example script is:

```
call with name "MBS_Memory_Set_Field" in dictionary 5261,
"Variable", 'Customer Number' of window RM_Customer_Maintenance of
form RM_Customer_Maintenance;
```

## MBS_Memory_Get

This call is used to read a previously stored value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

String data can be up to 132 characters per variable.

The parameter list for this call is:

```
in string IN_Parameter;
out anonymous field OUT_Value;
```

An example script is:

```
local string l_value;

call with name "MBS_Memory_Get" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Get_Boolean

This call is used to read a previously stored boolean value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Boolean Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
out boolean OUT_Value;
```

An example script is:

```
local boolean l_value;

call with name "MBS_Memory_Get" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Get_Currency

This call is used to read a previously stored currency value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Currency Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
out currency OUT_Value;
```

An example script is:

```
local currency l_value;

call with name "MBS_Memory_Get_Currency" in dictionary 5261,
"Variable", l_value;
```

## MBS_Memory_Get_Date

This call is used to read a previously stored date value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Date Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
out date OUT_Value;
```

An example script is:

```
local date l_value;

call with name "MBS_Memory_Get_Date" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Get_Long

This call is used to read a previously stored integer or long value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Long Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
out long OUT_Value;
```

An example script is:

```
local long l_value;

call with name "MBS_Memory_Get_Long" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Get_String

This call is used to read a previously stored string value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_String Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

String data can be up to 132 characters per variable.

The parameter list for this call is:

```
in string IN_Parameter;
out string OUT_Value;
```

An example script is:

```
local string l_value;

call with name "MBS_Memory_Get_String" in dictionary 5261,
"Variable", l_value;
```

## MBS_Memory_Get_Time

This call is used to read a previously stored time value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Time Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Parameter;
out time OUT_Value;
```

An example script is:

```
local time l_value;

call with name "MBS_Memory_Get_Time" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Get_Reference

This call is used to read a previously stored table or field reference from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Reference or MBS_Memory_Set_Table Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
out reference OUT_Reference;
```

An example script is:

```
local reference l_reference;
local string l_customer;

call with name "MBS_Memory_Get_Reference" in dictionary 5261,
"Variable", l_reference;
l_customer = column("Customer Number") of table(l_reference);
```

or

```
local reference l_reference;
local string l_customer;

call with name "MBS_Memory_Get_Reference" in dictionary 5261,
"Variable", l_reference;
l_customer = field(l_reference);
```

## MBS_Memory_Del

This call is used to remove a previously stored value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The value parameter for this helper function is only needed to identify the datatype.

The parameter list for this call is:

```
in string IN_Name;
in anonymous field IN_Value;
```

An example script is:

```
local string l_value;

call with name "MBS_Memory_Del" in dictionary 5261, "Variable",
l_value;
```

## MBS_Memory_Del_Boolean

This call is used to remove a previously stored boolean value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Boolean Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Boolean" in dictionary 5261,
"Variable";
```

## MBS_Memory_Del_Currency

This call is used to remove a previously stored currency value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Currency Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Currency" in dictionary 5261,
"Variable";
```

## MBS_Memory_Del_Date

This call is used to remove a previously stored date value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Date Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Date" in dictionary 5261, "Variable";
```

## MBS_Memory_Del_Long

This call is used to remove a previously stored integer or long value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Long Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Long" in dictionary 5261, "Variable";
```

## MBS_Memory_Del_String

This call is used to remove a previously stored string value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_String Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_String" in dictionary 5261,
"Variable";
```

## MBS_Memory_Del_Time

This call is used to remove a previously stored time value from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Time Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Time" in dictionary 5261, "Variable";
```

## MBS_Memory_Del_Reference

This call is used to remove a previously stored table or field reference from a memory variable slot. It is designed to be used with the MBS_Runtime_Execute and MBS_Memory_Set_Reference, MBS_Memory_Set_Table or MBS_Memory_Set_Field Helper Functions as a method of storing data for longer than the current script's scope or passing parameters. You can store up to 100 different variables of each datatype.

The parameter list for this call is:

```
in string IN_Name;
```

An example script is:

```
call with name "MBS_Memory_Del_Reference" in dictionary 5261,
"Variable";
```

## MBS_Get_Constant

This call is used read the value of a constant stored in a dictionary at the global or form level.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Constant_Name;
out anonymous field OUT_Constant_Value;
```

An example script is:

```
local integer l_constant;

call with name "MBS_Get_Constant" in dictionary 5261, 0, "",
"RM_DOC_SALES", l_constant;


call with name "MBS_Get_Constant" in dictionary 5261, 0,
"GL_Batch_Entry", "ORIGIN_GENERAL_ENTRY", l_constant;
```

## MBS_Get_Constant_Currency

This call is used read the value of a currency constant stored in a dictionary at the global or form level.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Constant_Name;
out currency OUT_Constant_Value;
```

An example script is:

```
local currency l_constant;

call with name "MBS_Get_Constant_Currency" in dictionary 5261, 0,
"", " MAX_QTY_5", l_constant;
```

## MBS_Get_Constant_Integer

This call is used read the value of an integer or long constant stored in a dictionary at the global or form level.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Constant_Name;
out long OUT_Constant_Value;
```

An example script is:

```
local long l_constant;

call with name "MBS_Get_Constant_Integer" in dictionary 5261, 0, "",
"EFT_SWITZERLAND", l_constant;
```

## MBS_Get_Constant_String

This call is used read the value of a string constant stored in a dictionary at the global or form level.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Constant_Name;
out string OUT_Constant_Value;
```

An example script is:

```
local string l_constant;

call with name "MBS_Get_Constant_String" in dictionary 5261, 0, "",
"SQL_DEFAULT_OWNER", l_constant;
```

## MBS_Set_Global

This call is used write the value of a global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in anonymous field IN_Global_Value;
```

An example script is:

```
local string l_global;

l_global = "TWO";
call with name "MBS_Set_Global" in dictionary 5261, 0, "Intercompany
ID", l_global;
```

## MBS_Set_Global_Boolean

This call is used write the value of a boolean global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in boolean IN_Global_Value;
```

An example script is:

```
local boolean l_global;

l_global = true;
call with name "MBS_Set_Global_Boolean" in dictionary 5261, 0, "Show
Status", l_global;
```

## MBS_Set_Global_Date

This call is used write the value of a date global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in date IN_Global_Value;
```

An example script is:

```
local date l_global;

l_global = sysdate();
call with name "MBS_Set_Global_Date" in dictionary 5261, 0, "User
Date", l_global;
```

## MBS_Set_Global_Numeric

This call is used write the value of a numeric global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in vcurrency IN_Global_Value;
```

An example script is:

```
local vcurrency l_global;

l_global = 1;
call with name "MBS_Set_Global_Numeric" in dictionary 5261, 0,
"Company ID", l_global;
```

## MBS_Set_Global_String

This call is used write the value of a string global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in string IN_Global_Value;
```

An example script is:

```
local string l_global;

l_global = "sa";
call with name "MBS_Set_Global_String" in dictionary 5261, 0, "User
ID", l_global;
```

## MBS_Set_Global_Text

This call is used write the value of a text global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in text IN_Global_Value;
```

An example script is:

```
local text l_global;

l_global = "";
call with name "MBS_Set_Global_Text" in dictionary 5261, 0, "Big
Text", l_global;
```

## MBS_Set_Global_Time

This call is used write the value of a time global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
in time IN_Global_Value;
```

An example script is:

```
local time l_global;

l_global = systime();
call with name "MBS_Set_Global_Time" in dictionary 5261, 0, "User
Time", l_global;
```

## MBS_Get_Global

This call is used read the value of a global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out anonymous field OUT_Global_Value;
```

An example script is:

```
local string l_global;

call with name "MBS_Get_Global" in dictionary 5261, 0, "Intercompany
ID", l_global;
```

## MBS_Get_Global_Boolean

This call is used read the value of a boolean global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out boolean OUT_Global_Value;
```

An example script is:

```
local boolean l_global;

call with name "MBS_Get_Global_Boolean" in dictionary 5261, 0, "Show
Status", l_global;
```

## MBS_Get_Global_Date

This call is used read the value of a date global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out date OUT_Global_Value;
```

An example script is:

```
local date l_global;

call with name "MBS_Get_Global_Date" in dictionary 5261, 0, "User
Date", l_global;
```

## MBS_Get_Global_Numeric

This call is used read the value of a numeric global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out vcurrency OUT_Global_Value;
```

An example script is:

```
local vcurrency l_global;

call with name "MBS_Get_Global_Numeric" in dictionary 5261, 0,
"Company ID", l_global;
```

## MBS_Get_Global_String

This call is used read the value of a string global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out string OUT_Global_Value;
```

An example script is:

```
local string l_global;

call with name "MBS_Get_Global_String" in dictionary 5261, 0, "User
ID", l_global;
```

## MBS_Get_Global_Text

This call is used read the value of a text global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out text OUT_Global_Value;
```

An example script is:

```
local text l_global;

call with name "MBS_Get_Global_Text" in dictionary 5261, 0, "Big
Text", l_global;
```

## MBS_Get_Global_Time

This call is used read the value of a time global variable stored in a dictionary.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Global_Name;
out time OUT_Global_Value;
```

An example script is:

```
local time l_global;

call with name "MBS_Get_Global_Time" in dictionary 5261, 0, "User
Time", l_global;
```

## MBS_Auto_Log

This call is used to add a message into the GP Power Tools log file. It is designed to be used with the Automatic Trigger Mode to record additional information when a trigger fires.

*Using this Helper Function within a Trigger Script will write to the GP Power Tools log file and also be recorded in the email body if the trigger is set to send an email.*

The parameter list for this call is:

```
in string IN_Message;
```

An example script is:

```
call with name "MBS_Auto_Log" in dictionary 5261, "Message";
```

## MBS_Logging_Start

This call is used to programmatically start Manual Logging Mode and is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

There are no parameters for this call.

An example script is:

```
call with name "MBS_Logging_Start" in dictionary 5261;
```

## MBS_Logging_Stop

This call is used to programmatically stop Manual Logging Mode and is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

There are no parameters for this call.

An example script is:

```
call with name "MBS_Logging_Stop" in dictionary 5261;
```

## MBS_Trigger_Start

This call is used to activate an Automatic Trigger Mode Trigger and is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_Start" in dictionary 5261, "XXXX";
```

## MBS_Trigger_Stop

This call is used to deactivate an Automatic Trigger Mode Trigger and is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_Stop" in dictionary 5261, "XXXX";
```

## MBS_Trigger_Update_Dialog

This call is used to override the default dialog message and dialog type on a trigger so it can be dynamically controlled from the script and is designed to be used with.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_Warning;
in integer IN_Mode; { 1 = info, 2 = warning, 3 = error, 4 = debug }
```

An example script is:

```
call with name "MBS_Trigger_Update_Dialog" in dictionary 5261,
"Message", 2 {warning};
```

## MBS_Trigger_Update_Email

This call is used to override the default email address on a trigger so it can be dynamically controlled from the script and is designed to be used with.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_Address;
```

An example script is:

```
call with name "MBS_Trigger_Update_Email" in dictionary 5261,
"user@domain.com";
```

## MBS_Trigger_Update_Email

This call is used to override the default email address on a trigger so it can be dynamically controlled from the script and is designed to be used with.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_Address;
```

An example script is:

```
call with name "MBS_Trigger_Update_Email" in dictionary 5261,
"user@domain.com";
```

## MBS_Arguments_Get_Count

This call is used to retrieve the number of Arguments (procedure or function parameters) available when using Global Level with Parameters or the Form Level with Parameters Trigger Events with Automatic Trigger Mode.

See full example in the MBS_Arguments_Get_Value helper function section.

The parameter list for this call is:

```
inout text INOUT_Args;
out integer OUT_Count;
```

An example script is:

```
inout text INOUT_Args;
local integer l_count;

call with name "MBS_Arguments_Get_Count" in dictionary 5261,
INOUT_Args, l_count;
```

## MBS_Arguments_Get_Type

This call is used to retrieve the datatype of a specified Argument (procedure or function parameter) available when using Global Level with Parameters or the Form Level with Parameters Trigger Events with Automatic Trigger Mode.

Types returned can be: boolean, integer, long, currency, vcurrency, date, time, datetime, string and text.

See full example in the MBS_Arguments_Get_Value helper function section.

The parameter list for this call is:

```
inout text INOUT_Args;
in integer IN_Position;
out string OUT_Type;
```

An example script is:

```
inout text INOUT_Args;
local string l_type;

call with name "MBS_Arguments_Get_Type" in dictionary 5261,
INOUT_Args, 1, l_type;
```

## MBS_Arguments_Get_Value

This call is used to retrieve the value of a specified Argument (procedure or function parameter) available when using Global Level with Parameters or the Form Level with Parameters Trigger Events with Automatic Trigger Mode.

Types can be: boolean, integer, long, currency, vcurrency, date, time, datetime, string and text.

The parameter list for this call is:

```
inout text INOUT_Args;
in integer IN_Position;
out anonymous field OUT_Value;
```

An example script is:

```
inout text INOUT_Args;
local string l_value;

call with name "MBS_Arguments_Get_Value" in dictionary 5261,
INOUT_Args, 1, l_value;
```

A more complex example of iterating through arguments is shown in the script below:

```
inout text INOUT_Args;
out boolean OUT_Condition;

local integer l_count;
local integer i;
local string l_type;
local string l_string;
local text l_text;
local boolean l_boolean;
local integer l_integer;
local long l_long;
local currency l_currency;
local vcurrency lvcurrency;
local date l_date;
local time l_time;
local datetime l_datetime;

OUT_Condition = false;

call with name "MBS_Arguments_Get_Count" in dictionary 5261, INOUT_Args, l_count;
if l_count > 0 then
    warning str(l_count);
    for i = 1 to l_count do
        call with name "MBS_Arguments_Get_Type" in dictionary 5261, INOUT_Args, i, l_type;
        case l_type
            in ["boolean"]
                call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                    i, l_boolean;
```

```
                    warning str(i)+": "+l_type+": " + str(l_boolean);
                in ["integer"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_integer;
                    warning str(i)+": "+l_type+": " + str(l_integer);
                in ["long"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_long;
                    warning str(i)+": "+l_type+": " + str(l_long);
                in ["currency"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_currency;
                    warning str(i)+": "+l_type+": " + str(l_currency);
                in ["vcurrency"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, lvcurrency;
                    warning str(i)+": "+l_type+": " + str(lvcurrency);
                in ["date"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_date;
                    warning str(i)+": "+l_type+": " + str(l_date);
                in ["time"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_time;
                    warning str(i)+": "+l_type+": " + str(l_time);
                in ["datetime", "date or time"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_datetime;
                    warning str(i)+": "+l_type+": " + str(l_datetime);
                in ["string"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_string;
                    warning str(i)+": "+l_type+": " + l_string;
                in ["text"]
                    call with name "MBS_Arguments_Get_Value" in dictionary 5261, INOUT_Args,
                        i, l_text;
                    warning str(i)+": "+l_type+": " + l_text;
                else
            end case;
        end for;
    end if;
    OUT_Condition = true;
```

## MBS_Arguments_Set_Value

This call is used to update the value of a specified Argument (procedure or function parameter) available when using Global Level with Parameters or the Form Level with Parameters Trigger Events with Automatic Trigger Mode.

Types can be: boolean, integer, long, currency, vcurrency, date, time, datetime, string and text.

The parameter list for this call is:

```
inout text INOUT_Args;
in integer IN_Position;
in anonymous field IN_Value;
```

An example script is:

```
inout text INOUT_Args;
local string l_value;

l_value = "string";
call with name "MBS_Arguments_Set_Value" in dictionary 5261,
INOUT_Args, 1, l_value;
```

## MBS_DUOS_Set

This call is used to store a value in the DUOS SY_User_Object_Store (SY90000) . It is designed to be used with the MBS_DUOS_Get Helper Function.

The parameter list for this call is:

```
in string IN_Object;
in string IN_ID;
in string IN_Property;
in string IN_Value;
```

An example script is:

```
local string l_string;
local string l_object;

l_object = "ID";
l_string = "Value";
call with name "MBS_DUOS_Set" in dictionary 5261, "Object",
l_object, "Property", l_string;
```

## MBS_DUOS_Get

This call is used to read a previously set value from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_DUOS_Set Helper Function.

The parameter list for this call is:

```
in string IN_Object;
in string IN_ID;
in string IN_Property;
out string OUT_Value;
```

An example script is:

```
local string l_string;
local string l_object;

l_object = "ID";
call with name "MBS_DUOS_Get" in dictionary 5261, "Object",
l_object, "Property", l_string;
```

## MBS_DUOS_Del

This call is used to remove a previously set value from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_DUOS_Set Helper Function.

The parameter list for this call is:

```
in string IN_Object;
in string IN_ID;
in string IN_Property;
```

An example script is:

```
local string l_object;

l_object = "ID";
call with name "MBS_DUOS_Del" in dictionary 5261, "Object",
l_object, "Property";
```

## MBS_DUOS_DelAll

This call is used to remove all previously stored values for an object from the DUOS SY_User_Object_Store (SY90000) table. It is designed to be used with the MBS_DUOS_Set Helper Function.

The parameter list for this call is:

```
in string IN_Object;
in string IN_ID;
```

An example script is:

```
l_object = "ID";
call with name "MBS_DUOS_DelAll" in dictionary 5261, "Object",
l_object;
```

## MBS_UserAddInfo_Get

This call is used to retrieve data stored in the User Setup Additional Information window.

The parameter list for this call is:

```
in 'User ID' IN_UserID; {User ID}
in integer IN_Position; {which field to return: Field 1 to 20}
out string OUT_String; {Returned Data}
```

An example script is:

```
local string l_user;
local string l_string;

l_user = 'User ID' of globals;
call with name " MBS_UserAddInfo_Get" in dictionary 5261, l_user, 1
{field to get}, l_string;
```

## MBS_UserAddInfo_Set

This call is used to update data stored in the User Setup Additional Information window.

The parameter list for this call is:

```
in 'User ID' IN_UserID; {User ID}
in integer IN_Position; {which field to return: Field 1 to 20}
in string IN_String; {Data to update}
```

An example script is:

```
local string l_user;
local string l_string;

l_user = 'User ID' of globals;
l_string = "email@domain.com";
call with name " MBS_UserAddInfo_Set" in dictionary 5261, l_user, 1
{field to set}, l_string;
```

## MBS_UserAddInfo_GetPrompt

This call is used to retrieve prompts used for the User defined fields in the User Setup Additional Information window.

The parameter list for this call is:

```
in integer IN_Position; {which field to return: Field 1 to 6}
out string OUT_String; {Returned Prompt}
```

An example script is:

```
local string l_prompt;

call with name " MBS_UserAddInfo_GetPrompt" in dictionary 5261, 1
{prompt to get}, l_prompt;
```

## MBS_SQL_Lookup

This call is used to open a lookup and return the selected value to a field. It uses the Custom SQL Lookup from Parameter Lists. You need to provide a SQL Execute Setup script which returns a query with three string columns; an ID string, a Description string and a string to be returned (usually the same as the ID value). The other parameters are the seed value and return field (usually the same window field).

Return does not work across dictionaries, use the MBS_SQL_Lookup2 helper function to set field and run script field instead.

The parameter list for this call is:

```
in string IN_Script_ID;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_SQL_Lookup" in dictionary 5261, "XXXX",
l_string, Return_Field;
```

## MBS_SQL_Lookup2

This call is used to open a lookup and return the selected value to a field. It uses the Custom SQL Lookup from Parameter Lists. You need to provide a SQL Execute Setup script which returns a query with three string columns; an ID string, a Description string and a string to be returned (usually the same as the ID value). The other parameters are the seed value and return field (usually the same window field).

This helper function uses set field and run script field and can be used when the MBS_SQL_Lookup helper function does not work.

The parameter list for this call is:

```
in string IN_Script_ID;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_SQL_Lookup2" in dictionary 5261, "XXXX",
l_string, Return_Field;
```

## MBS_SQL_Lookup_Parameter

This call is used to open a lookup and return the selected value to a field. It uses the Custom SQL Lookup from Parameter Lists. You need to provide a Parameter List Parameter ID and the Position for parameter that is set up for a Custom Lookup (SQL) mode. The other parameters are the seed value and return field (usually the same window field).

Return does not work across dictionaries, use the MBS_SQL_Lookup_Parameter2 helper function to set field and run script field instead.

The parameter list for this call is:

```
in string IN_Parameter_ID;
in integer IN_Position;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_SQL_Lookup_Parameter" in dictionary 5261,
"XXXX", Y, l_string, Return_Field;
```

## MBS_SQL_Lookup_Parameter2

This call is used to open a lookup and return the selected value to a field. It uses the Custom SQL Lookup from Parameter Lists. You need to provide a Parameter List Parameter ID and the Position for parameter that is set up for a Custom Lookup (SQL) mode. The other parameters are the seed value and return field (usually the same window field).

This helper function uses set field and run script field and can be used when the MBS_SQL_Lookup_Parameter helper function does not work.

The parameter list for this call is:

```
in string IN_Parameter_ID;
in integer IN_Position;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_SQL_Lookup_Parameter2" in dictionary 5261,
"XXXX", Y, l_string, Return_Field;
```

## MBS_SQL_Lookup_Validate

This call is used to validate data against a Custom SQL Lookup used by Parameter Lists. You need to provide a SQL Execute Setup script which returns a query with three string columns; an ID string, a Description string and a string to be returned (usually the same as the ID value). The other parameters are the value and returned boolean result.

The parameter list for this call is:

```
in string IN_Script_ID;
in string IN_Value;
out boolean OUT_Validated;
```

An example script is:

```
local string l_string;
local boolean l_validated;

call with name "MBS_SQL_Lookup_Validate" in dictionary 5261, "XXXX",
l_string, l_validated;
```

## MBS_SQL_Lookup_Parameter_Validate

This call is used to validate data against a Custom SQL Lookup used by Parameter Lists. You need to provide a Parameter List Parameter ID and the Position for parameter that is set up for a Custom Lookup (SQL) mode. The other parameters are the value and returned boolean result.

The parameter list for this call is:

```
in string IN_Parameter_ID;
in integer IN_Position;
in string IN_Value;
out boolean OUT_Validated;
```

An example script is:

```
local string l_string;
local boolean l_validated;

call with name "MBS_SQL_Lookup_Parameter_Validate" in dictionary
5261, "XXXX", Y, l_string, l_validated;
```

## MBS_Form_Lookup

This call is used to open a lookup and return the selected value to a field. It uses the Custom Form Lookup from Parameter Lists. You need to define the form, window and field information required to drive an existing lookup form in any dictionary installed in Microsoft Dynamics GP. The other parameters are the seed value and return field (usually the same window field).

Return does not work across dictionaries, use the MBS_Form_Lookup2 helper function to set field and run script field instead.

The parameter list for this call is:

```
in integer IN_Dict;
in string IN_Form;
in string IN_Window;
in string IN_Field;
in string IN_FieldSortBy;
in string IN_WindowScroll;
in string IN_FieldScroll;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_Form_Lookup" in dictionary 5261, Dict, Form,
Window, Field, FieldSortBy, WindowScroll, FieldScroll, l_string,
Return_Field;
```

## MBS_Form_Lookup2

This call is used to open a lookup and return the selected value to a field. It uses the Custom Form Lookup from Parameter Lists. You need to define the form, window and field information required to drive an existing lookup form in any dictionary installed in Microsoft Dynamics GP. The other parameters are the seed value and return field (usually the same window field).

This helper function uses set field and run script field and can be used when the MBS_Form_Lookup helper function does not work.

The parameter list for this call is:

```
in integer IN_Dict;
in string IN_Form;
in string IN_Window;
in string IN_Field;
in string IN_FieldSortBy;
in string IN_WindowScroll;
in string IN_FieldScroll;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_Form_Lookup2" in dictionary 5261, Dict, Form,
Window, Field, FieldSortBy, WindowScroll, FieldScroll, l_string,
Return_Field;
```

## MBS_Form_Lookup_Parameter

This call is used to open a lookup and return the selected value to a field. It uses the Custom Form Lookup from Parameter Lists. You need to provide a Parameter List Parameter ID and the Position for parameter that is set up for a Custom Lookup (Form) mode. The other parameters are the seed value and return field (usually the same window field).

Return does not work across dictionaries, use the MBS_Form_Lookup_Parameter2 helper function to set field and run script field instead.

The parameter list for this call is:

```
in string IN_Parameter_ID;
in integer IN_Position;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_Form_Lookup_Parameter" in dictionary 5261,
"XXXX", Y, l_string, Return_Field;
```

## MBS_Form_Lookup_Parameter2

This call is used to open a lookup and return the selected value to a field. It uses the Custom Form Lookup from Parameter Lists. You need to provide a Parameter List Parameter ID and the Position for parameter that is set up for a Custom Lookup (Form) mode. The other parameters are the seed value and return field (usually the same window field).

This helper function uses set field and run script field and can be used when the MBS_Form_Lookup_Parameter helper function does not work.

The parameter list for this call is:

```
in string IN_Parameter_ID;
in integer IN_Position;
in string IN_Seed_Value;
inout anonymous INOUT_Return_Field;
```

An example script is:

```
local string l_string;

call with name "MBS_Form_Lookup_Parameter2" in dictionary 5261,
"XXXX", Y, l_string, Return_Field;
```

## MBS_Project_Start

This call is used to activate all Automatic Trigger Mode Triggers belonging to the specified Project ID. It is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_ProjectID;
```

An example script is:

```
call with name "MBS_Project_Start" in dictionary 5261, "XXXX";
```

## MBS_Project_Stop

This call is used to deactivate all Automatic Trigger Mode Triggers belonging to the specified Project ID. It is designed to be used with Non-logging triggers.in the Automatic Trigger Mode.

The parameter list for this call is:

```
in string IN_ProjectID;
```

An example script is:

```
call with name "MBS_Project_Stop" in dictionary 5261, "XXXX";
```

## MBS_Script_Substitute

Use this call to replace all instances of a placeholder in a script with a value in script. It can be used to manually perform Parameter List substitution. Use this function with the MBS_Parameter_String function and MBS_Parameter_String, MBS_Parameter_Number, MBS_Parameter_Currency, MBS_Parameter_Boolean, MBS_Parameter_Date and MBS_Parameter_Time functions.

The parameter list for this call is:

```
inout text INOUT_Text;
in string IN_Placeholder;
in string IN_Value;
```

An example script is:

```
local text MBS_Text_Field;
local string MBS_Placeholder;
local string MBS_Value;

call with name "MBS_Script_Substitute" in dictionary 5261,
    MBS_Text_Field, MBS_Placeholder, MBS_Value;
```

## MBS_Parameter_Placeholder

Use this call to obtain the Parameter List placeholder to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in integer IN_Type; { 1 = String, 2 = Integer, 3 = Currency, 4 =
Boolean, 5 = Date, 6 = Time }
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Placeholder;


call with name "MBS_Parameter_Placeholder" in dictionary 5261,
    1 {Type: String}, 1 {Position}, 0 {FromTo: From/Single}, 1
{Language: Dexterity sanScript}, MBS_Placeholder;
```

## MBS_Parameter_String

Use this call to obtain the string representation of a string value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in string IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local string MBS_Value_String;

MBS_Value_String = <Variable>;
call with name "MBS_Parameter_String" in dictionary 5261,
    MBS_Value_String, 1 {Language: Dexterity sanScript}, MBS_Value;
```

## MBS_Parameter_Number

Use this call to obtain the string representation of a number value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in long IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local long MBS_Value_Number;

MBS_Value_Number = <Variable>;
call with name "MBS_Parameter_Number" in dictionary 5261,
    MBS_Value_Number, 1 {Language: Dexterity sanScript}, MBS_Value;
```

## MBS_Parameter_Currency

Use this call to obtain the string representation of a currency value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in currency IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local currency MBS_Value_Currency;

MBS_Value_Currency = <Variable>;
call with name "MBS_Parameter_Currency" in dictionary 5261,
    MBS_Value_Currency, 1 {Language: Dexterity sanScript},
MBS_Value;
```

## MBS_Parameter_Boolean

Use this call to obtain the string representation of a boolean value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in boolean IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local boolean MBS_Value_Boolean;

MBS_Value_Boolean = <Variable>;
call with name "MBS_Parameter_Boolean" in dictionary 5261,
    MBS_Value_Boolean, 1 {Language: Dexterity sanScript}, MBS_Value;
```

## MBS_Parameter_Date

Use this call to obtain the string representation of a date value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in date IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local date MBS_Value_Date;

MBS_Value_Date = <Variable>;
call with name "MBS_Parameter_Date" in dictionary 5261,
    MBS_Value_Date, 1 {Language: Dexterity sanScript}, MBS_Value;
```

## MBS_Parameter_Time

Use this call to obtain the string representation of a time value to use with the MBS_Script_Substitute function.

The parameter list for this call is:

```
in time IN_Value;
in integer IN_Language; { 1 = Dex, 2 = SQL, 3 = C#, 4 = VB }
out string OUT_String;
```

An example script is:

```
local string MBS_Value;
local time MBS_Value_Time;

MBS_Value_Time = <Variable>;
call with name "MBS_Parameter_Time" in dictionary 5261,
    MBS_Value_Time, 1 {Language: Dexterity sanScript}, MBS_Value;
```

## MBS_Parameter_Load

Use this call to a Parameter List Parameter ID with its default values. Use this command before using the MBS_Parameter_Set_String, MBS_Parameter_Set_Number, MBS_Parameter_Set_Currency, MBS_Parameter_Set_Boolean, MBS_Parameter_Set_Date, MBS_Parameter_Set_Time,. MBS_Parameter_Get_String, MBS_Parameter_Get_Number, MBS_Parameter_Get_Currency, MBS_Parameter_Get_Boolean, MBS_Parameter_Get_Date, and MBS_Parameter_Get_Time functions

The parameter list for this call is:

```
in string IN_ParameterID;
out integer OUT_Status;
```

An example script is:

```
local integer MBS_Status;

call with name "MBS_Parameter_Load" in dictionary 5261, "XXXX",
MBS_Status;
```

## MBS_Parameter_Open

Use this call to open the Parameter List window to a previously loaded Parameter List and specify the Runtime Execute Script ID to be executed when the Parameter List window's OK Button is clicked.

The parameter list for this call is:

```
in string IN_ScriptID;
out integer OUT_Status;
```

An example script is:

```
local integer MBS_Status;

call with name "MBS_Parameter_Open" in dictionary 5261, "XXXX",
MBS_Status;
```

## MBS_Parameter_Set_String

Use this call to set the value of a string parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in string IN_Value;
```

An example script is:

```
local string MBS_Value_String;

MBS_Value_String = <Variable>;
call with name "MBS_Parameter_Set_String" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_String;
```

## MBS_Parameter_Set_Number

Use this call to set the value of a number parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in long IN_Value;
```

An example script is:

```
local long MBS_Value_Number;

MBS_Value_Number = <Variable>;
call with name "MBS_Parameter_Set_Number" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Number;
```

## MBS_Parameter_Set_Currency

Use this call to set the value of a currency parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in currency IN_Value;
```

An example script is:

```
local currency MBS_Value_Currency;

MBS_Value_Currency = <Variable>;
call with name "MBS_Parameter_Set_Currency" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single},
MBS_Value_Currency;
```

## MBS_Parameter_Set_Boolean

Use this call to set the value of a boolean parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in boolean IN_Value;
```

An example script is:

```
local boolean MBS_Value_Boolean;

MBS_Value_Boolean = <Variable>;
call with name "MBS_Parameter_Set_Boolean" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single},
MBS_Value_Boolean;
```

## MBS_Parameter_Set_Date

Use this call to set the value of a date parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in date IN_Value;
```

An example script is:

```
local date MBS_Value_Date;

MBS_Value_Date = <Variable>;
call with name "MBS_Parameter_Set_Date" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Date;
```

## MBS_Parameter_Set_Time

Use this call to set the value of a time parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
in time IN_Value;
```

An example script is:

```
local time MBS_Value_Time;

MBS_Value_Time = <Variable>;
call with name "MBS_Parameter_Set_Time" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Time;
```

## MBS_Parameter_Get_String

Use this call to get the value of a string parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out string OUT_Value;
```

An example script is:

```
local string MBS_Value_String;

call with name "MBS_Parameter_Get_String" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_String;
```

## MBS_Parameter_Get_Number

Use this call to get the value of a number parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out long OUT_Value;
```

An example script is:

```
local long MBS_Value_Number;

call with name "MBS_Parameter_Get_Number" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Number;
```

## MBS_Parameter_Get_Currency

Use this call to get the value of a currency parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out currency OUT_Value;
```

An example script is:

```
local currency MBS_Value_Currency;

call with name "MBS_Parameter_Get_Currency" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single},
MBS_Value_Currency;
```

## MBS_Parameter_Get_Boolean

Use this call to get the value of a boolean parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out boolean OUT_Value;
```

An example script is:

```
local boolean MBS_Value_Boolean;

call with name "MBS_Parameter_Get_Boolean" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single},
MBS_Value_Boolean;
```

## MBS_Parameter_Get_Date

Use this call to get the value of a date parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out date OUT_Value;
```

An example script is:

```
local date MBS_Value_Date;

call with name "MBS_Parameter_Get_Date" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Date;
```

## MBS_Parameter_Get_Time

Use this call to get the value of a time parameter into the parameter list memory. The MBS_Parameter_Load must be executed before using this function to initialize the parameter list in memory.

The parameter list for this call is:

```
in integer IN_Position;
in integer IN_From; { 0 = From, 1 = To }
out time OUT_Value;
```

An example script is:

```
local time MBS_Value_Time;

call with name "MBS_Parameter_Get_Time" in dictionary 5261,
    1 {Position: "XXXX"}, 0 {FromTo: From/Single}, MBS_Value_Time;
```

## MBS_Convert

Use this call to convert a string representation of data to any datatype.

The parameter list for this call is:

```
in text IN_Text;
out anonymous field OUT_Data;
```

An example script is:

```
local text l_text;
local string l_data;
call with name "MBS_Convert " in dictionary 5261, l_text, l_data;
```

## MBS_Convert_Boolean

Use this call to convert a string representation of data to a Boolean datatype.

The parameter list for this call is:

```
in text IN_Text;
out boolean OUT_Boolean;
```

An example script is:

```
local text l_text;
local boolean l_data;
call with name "MBS_Convert_Boolean" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Currency

Use this call to convert a string representation of data to a Currency datatype.

The parameter list for this call is:

```
in text IN_Text;
out currency OUT_Currency;
```

An example script is:

```
local text l_text;
local currency l_data;
call with name "MBS_Convert_Currency" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Date

Use this call to convert a string representation of data to a Date datatype.

The parameter list for this call is:

```
in text IN_Text;
out date OUT_Date;
```

An example script is:

```
local text l_text;
local date l_data;
call with name "MBS_Convert_Date" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Datetime

Use this call to convert a string representation of data to a Datetime datatype.

The parameter list for this call is:

```
in text IN_Text;
out datetime OUT_Datetime;
```

An example script is:

```
local text l_text;
local datetime l_data;
call with name "MBS_Convert_Datetime" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Integer

Use this call to convert a string representation of data to an Integer datatype.

The parameter list for this call is:

```
in text IN_Text;
out integer OUT_Integer;
```

An example script is:

```
local text l_text;
local integer l_data;
call with name "MBS_Convert_Integer" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Long

Use this call to convert a string representation of data to a Long datatype.

The parameter list for this call is:

```
in text IN_Text;
out long OUT_Long;
```

An example script is:

```
local text l_text;
local long l_data;
call with name "MBS_Convert_Long" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_String

Use this call to convert a string representation of data to a String datatype.

The parameter list for this call is:

```
in text IN_Text;
out string OUT_String;
```

An example script is:

```
local text l_text;
local string l_data;
call with name "MBS_Convert_String" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Text

Use this call to convert a string representation of data to a Text datatype.

The parameter list for this call is:

```
in text IN_Text;
out text OUT_Text;
```

An example script is:

```
local text l_text;
local text l_data;
call with name "MBS_Convert_Text" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_Time

Use this call to convert a string representation of data to a Time datatype.

The parameter list for this call is:

```
in text IN_Text;
out time OUT_Time;
```

An example script is:

```
local text l_text;
local time l_data;
call with name "MBS_Convert_Time" in dictionary 5261, l_text,
l_data;
```

## MBS_Convert_VCurrency

Use this call to convert a string representation of data to a VCurrency datatype.

The parameter list for this call is:

```
in text IN_Text;
out vcurrency OUT_VCurrency;
```

An example script is:

```
local text l_text;
local vcurrency l_data;
call with name "MBS_Convert_VCurrency" in dictionary 5261, l_text,
l_data;
```

## MBS_Return_By_Field

Use this call to return data to a field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Return does not work across dictionaries, use the MBS_Return_By_Field2 helper function to set field and run script field instead.

The parameter list for this call is:

```
inout anonymous field INOUT_Field;
in anonymous IN_Value;
in boolean IN_Nofocus;
in boolean IN_Delayed;
in boolean IN_Forced;
```

An example script is:

```
call with name "MBS_Return_By_Field" in dictionary 5261, <Field>,
    <Value>,  false {No Focus}, false {Delayed}, false {Forced};
```

## MBS_Return_By_Field2

Use this call to return data to a field. It will not work properly if the field's change script contains old() or diff() functions, in this case look at the MBS_Map_By_Field helper function instead.

This helper function uses set field and run script field and can be used when the MBS_Return_By_Field helper function does not work.

The parameter list for this call is:

```
inout anonymous field INOUT_Field;
in anonymous IN_Value;
in boolean IN_Nofocus;
in boolean IN_Delayed;
in boolean IN_Forced;
```

An example script is:

```
call with name "MBS_Return_By_Field2" in dictionary 5261, <Field>,
    <Value>,  false {No Focus}, false {Delayed}, false {Forced};
```

## MBS_Return_By_Reference

Use this call to return data to a field by reference. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Return does not work across dictionaries, use the MBS_Return_By_Reference2 helper function to set field and run script field instead.

The parameter list for this call is:

```
inout reference INOUT_Field_Reference;
in anonymous IN_Value;
in boolean IN_Nofocus;
in boolean IN_Delayed;
in boolean IN_Forced;
```

An example script is:

```
call with name "MBS_Return_By_Reference" in dictionary 5261, <Field
Reference>,
    <Value>,  false {No Focus}, false {Delayed}, false {Forced};
```

## MBS_Return_By_Reference2

Use this call to return data to a field by reference. It will not work properly if the field's change script contains old() or diff() functions, in this case look at the MBS_Map_By_Reference helper function instead.

This helper function uses set field and run script field and can be used when the MBS_Return_By_Reference helper function does not work.

The parameter list for this call is:

```
inout reference INOUT_Field_Reference;
in anonymous IN_Value;
in boolean IN_Nofocus;
in boolean IN_Delayed;
in boolean IN_Forced;
```

An example script is:

```
call with name "MBS_Return_By_Reference2" in dictionary 5261, <Field
Reference>,
    <Value>,  false {No Focus}, false {Delayed}, false {Forced};
```

## MBS_Map_By_Field

Use this call to map data to a field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
inout anonymous field INOUT_Field;
in anonymous IN_Value;
```

An example script is:

```
call with name "MBS_Map_By_Field" in dictionary 5261, <Field>,
    <Value>;
```

## MBS_Map_By_Reference

Use this call to map data to a field by reference. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
inout reference INOUT_Field_Reference;
in anonymous IN_Value;
```

An example script is:

```
call with name "MBS_Map_By_Reference" in dictionary 5261, <Field>,
    <Value>;
```

## MBS_Map

Use this call to map data to a field of any datatype. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
inout anonymous field IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Field";

call with name "MBS_Map" in dictionary 5261, Dictionary, "Form",
"Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Boolean

Use this call to map data to a Boolean field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in boolean IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local boolean l_field;
l_field = false;

call with name "MBS_Map_Boolean" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Date

Use this call to map data to a Date field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in date IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local date l_field;
l_field = sysdate();

call with name "MBS_Map_Date" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Numeric

Use this call to map data to a Numeric field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in vcurrency IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local vcurrency l_field;
l_field = 0;

call with name "MBS_Map_Numeric" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_String

Use this call to map data to a String field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in string IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_field;
l_field = "Field";

call with name "MBS_Map_String" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Text

Use this call to map data to a Text field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in text IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local text l_field;
l_field = "Field";

call with name "MBS_Map_Text" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Time

Use this call to map data to a Time field. Using this call can get around issues caused when the field's change script contains old() or diff() functions when setting the field and running the script does not work as expected.

Use Window_PullFocus() command or MBS_Pull_Window_Focus Helper Function to force final field's change script to execute.

The parameter list for this call is:

```
in integer IN_Prod_ID;
in string IN_Form_Name;
in string IN_Window_Name;
in string IN_Field_Name;
in time IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local time l_field;
l_field = systime();

call with name "MBS_Map_Time" in dictionary 5261, Dictionary,
"Form", "Window", "Field", l_field, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Message

Use this call to retrieve the message text for a Message ID for the current language (or default language if no language specific message text available. To setup Messages use the Messages Setup window.

The parameter list for this call is:

```
in integer IN_Message;
out string OUT_Message;
```

An example script is:

```
local string l_message;

call with name "MBS_Get_Message" in dictionary 5261, "<Message ID>",
    l_message;
warning l_message;
```

## MBS_Get_Message_Prompts

Use this call to retrieve the message text and prompts for a Message ID for the current language (or default language if no language specific message text available. To setup Messages use the Messages Setup window.

The parameter list for this call is:

```
in integer IN_Message;
out string OUT_Message;
out string OUT_Prompt1;
out string OUT_Prompt2;
out string OUT_Prompt3;
```

An example script is:

```
local string l_message, l_prompt1, l_prompt2, l_prompt3;

call with name "MBS_Get_Message_Prompts" in dictionary 5261,
"<Message ID>",
    l_message, l_prompt1, l_prompt2, l_prompt3;
if ask(l_message, l_prompt1, l_prompt2, l_prompt3)= ASKBUTTON1 then

end if;
```

## MBS_getmsg

Use this call to get a message resource from any installed dictionary.

The parameter list for this call is:

```
in integer IN_DictID;
in integer IN_ID;
out string OUT_Message;
```

An example script is:

```
local string l_message;

call with name "MBS_getmsg" in dictionary 5261, 0, 1765,
    l_message;
warning l_message;
```

## MBS_Get_Error_Message

Use this call to get a system Error Message from the SY_Error_Messages_MSTR table or SY_Error_Messages_MSTR_3rd table (if the Error Number >= 100000).

The parameter list for this call is:

```
in long IN_Error_Number;
out string OUT_Message;
```

An example script is:

```
local string l_message;

call with name "MBS_Get_Error_Message" in dictionary 5261, 1,
    l_message;
warning l_message;
```

## MBS_Show_Dialog

Use this call to display a system dialog of the selected mode.

The parameter list for this call is:

```
in string IN_Message;
in integer IN_Mode; {1 - Info, 2 - Warning, 3 - Error, 4 - Debug}
```

An example script is:

```
local string l_message;

call with name " MBS_Show_Dialog" in dictionary 5261, 1,
    l_message, 1;
```

## MBS_Show_Dialog_Text

Use this call to display a system text dialog of the selected mode. The text can contain tabs, char(9), and carriage returns, char(13).

The parameter list for this call is:

```
in text IN_Message_Text;
in integer IN_Mode; {1 - Info, 2 - Warning, 3 - Error, 4 - Debug}
```

An example script is:

```
local text l_message_text;

call with name " MBS_Show_Dialog" in dictionary 5261, 1,
    l_message_text, 1;
```

## MBS_Ask_Dialog

Use this call to display an ask() system dialog with the selected buttons. It works well with the MBS_Get_Message_Prompts helper function.

The parameter list for this call is:

```
in string IN_Message;
in string IN_Prompt1;
in string IN_Prompt2;
in string IN_Prompt3;
out integer OUT_Response;
```

An example script is:

```
local string l_message, l_prompt1, l_prompt2, l_prompt3;
local integer l_response;

call with name "MBS_Ask_Dialog" in dictionary 5261, l_message,
    l_prompt1, l_prompt2, l_prompt3, l_response;
```

## MBS_Ask_Dialog_Text

Use this call to display an ask() text system dialog with the selected buttons. It works well with the MBS_Get_Message_Prompts helper function. The text can contain tabs, char(9), and carriage returns, char(13).

The parameter list for this call is:

```
in text IN_Message_Text;
in string IN_Prompt1;
in string IN_Prompt2;
in string IN_Prompt3;
out integer OUT_Response;
```

An example script is:

```
local text l_message_text, l_prompt1, l_prompt2, l_prompt3;
local integer l_response;

call with name "MBS_Ask_Dialog" in dictionary 5261, l_message_text,
    l_prompt1, l_prompt2, l_prompt3, l_response;
```

## MBS_Get_DateTime

Use this call to retrieve the current datetime value with date, time and milliseconds.

The parameter list for this call is:

```
in boolean IN_UTCTime;
out date OUT_Date;
out time OUT_Time;
out integer OUT_Milliseconds;
```

An example script is:

```
local date l_date;
local time l_time;
local integer l_milliseconds;


call with name "MBS_Get_DateTime" in dictionary 5261, true {UTC},
l_date, l_time, l_milliseconds;
```

## MBS_Token

Use this call break a character separated string into individual string values.

The parameter list for this call is:

```
in string IN_string;
in string IN_token;
in integer IN_position;
out string OUT_string;
```

An example script is:

```
local string l_parameters;
local string l_value;

call with name "MBS_Token" in dictionary 5261,
    l_parameters, CH_COMMA, 1, l_value;
```

## MBS_Field_ParseText

Use this call to break a text field into lines, similar to Field_ParseText() Dexterity function, but can return text variables longer than 255 characters.

The parameter list for this call is:

```
in text IN_Text;
in integer IN_Characters;
inout integer INOUT_Position;
out text OUT_Text;
```

An example script is:

```
inout text MBS_InText;
local integer MBS_StartPos;
local text MBS_Text;

repeat
    call with name "MBS_Field_ParseText" in dictionary 5261,
        MBS_InText, 32767, MBS_StartPos, MBS_Text;

until MBS_StartPos = 0;
```

## MBS_subtext

Use this call to return part of a text field, similar to substring() Dexterity function, but can return text variables longer than 255 characters.

The parameter list for this call is:

```
in text IN_Text;
out text OUT_Text;
```

An example script is:

```
inout text MBS_InText;
local text MBS_Text;

call with name "MBS_subtext" in dictionary 5261,
    MBS_InText, 1 {Start}, 300 {Length}, MBS_Text;
```

## MBS_Security_Form_Check

Use this call to manually check security to identify if additional fields are available on an alternate or modified window.

The parameter list for this call is:

```
in integer dictid;
in string resname;
out boolean access;
out integer altdictid;
out boolean modified;
```

An example script is:

```
local boolean access, modified;
local integer altdictid;

call with name "MBS_Security_Form_Check" in dictionary 5261,
    DYNAMICS, "PM_Vendor_Maintenance", access, altdictid, modified;
if access and altdictid = DYNAMICS and modified then

end if;
```

## MBS_Trigger_Disable

Use this call to programmatically update the Trigger Setup table to mark a trigger as disabled.

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_Disable" in dictionary 5261, "XXXX";
```

## MBS_Trigger_Enable

Use this call to programmatically update the Trigger Setup table to mark a trigger as enabled (not disabled).

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_Enable" in dictionary 5261, "XXXX";
```

## MBS_Trigger_DisableSingle

Use this call temporarily disable an already started trigger.

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_DisableSingle" in dictionary 5261,
"XXXX";
```

## MBS_Trigger_EnableSingle

Use this call re-enable a temporarily disabled trigger.

The parameter list for this call is:

```
in string IN_TriggerID;
```

An example script is:

```
call with name "MBS_Trigger_EnableSingle" in dictionary 5261,
"XXXX";
```

## MBS_Is_Trigger_Started

Use this call check if a trigger is already started.

The parameter list for this call is:

```
in string IN_TriggerID;
out boolean OUT_Started;
```

An example script is:

```
local booelan l_started;

call with name "MBS_Is_Trigger_Started" in dictionary 5261, "XXXX",
l_started;
```

## MBS_Is_Trigger_Enabled

Use this call check if a trigger is temporarily disabled or not.

The parameter list for this call is:

```
in string IN_TriggerID;
out boolean OUT_Enabled;
```

An example script is:

```
local booelan l_enabled;

call with name "MBS_Is_Trigger_Enabled" in dictionary 5261, "XXXX",
l_enabled;
```

## MBS_Exit_After_Processes

Use this call to request Microsoft Dynamics GP to exit once all background processes have been completed.

The parameter list for this call is:

```
<None>
```

An example script is:

```
call with name "MBS_Exit_After_Processes" in dictionary 5261;
```

## MBS_Switch_Company

Use this call to request Microsoft Dynamics GP to switch to a different company once all background processes are completed.

It can accept the company by its numeric Company ID, or by the Intercompany ID (Database name). If the company is left blank it will move to the next company in the drop down list.

There is an option to exit GP if it fails to identify the next company to move to.

The parameter list for this call is:

```
in string IN_Company;
in boolean IN_Exit_on_Fail;
```

An example script is:

```
call with name "MBS_Switch_Company" in dictionary 5261, " <Blank for
next, Company ID or Database>", <true/false: Exit if Fail>;
```

## MBS_CompanyColorGetRGB

Use this call to retrieve the RGB color data for the current GP Power Tools Company Color Theme.

It can be used to allow a .Net window to be colored to match the Dexterity windows.

The modes (1-5) are:

1. Window Toolbar
2. Window Background (commonly used)
3. Window Heading
4. Field Background (commonly used)
5. Scrolling Window Line

The parameter list for this call is:

```
in integer IN_Mode;
out integer OUT_red_level, OUT_green_level, OUT_blue_level;
```

An example script is:

```
local integer redColor;
local integer greenColor;
local integer blueColor;
call with name "MBS_CompanyColorGetRGB" in dictionary 5261, 2,
redColor, greenColor, blueColor;
```

Below is an excerpt of an example of how the call can be used with the .Net Interop in Dexterity to set colors on a dynamically created dialog:

```
{ Color Coding - Using function in GP Power Tools Build 29 or later }
buttonOk.BackColor = Color.FromArgb(253, 253, 253);
buttonCancel.BackColor = Color.FromArgb(253, 253, 253);
call with name "MBS_CompanyColorGetRGB" in dictionary 5261, 2, redColor,
greenColor, blueColor;
if (redColor + greenColor + blueColor) > 0 then
    l_form.BackColor = Color.FromArgb(redColor, greenColor, blueColor);
end if;
call with name "MBS_CompanyColorGetRGB" in dictionary 5261, 4, redColor,
greenColor, blueColor;
if (redColor + greenColor + blueColor) > 0 then
    textBox.BackColor = Color.FromArgb(redColor, greenColor, blueColor);
end if;
```

## MBS_Copy_To_Clipboard

Use this call to place text data onto the windows clipboard.

The parameter list for this call is:

```
in text IN_Text;
```

An example script is:

```
call with name "MBS_Copy_To_Clipboard" in dictionary 5261, "Text";
```

## MBS_Copy_From_Clipboard

Use this call to retrieve text data from the windows clipboard.

It can be used to retrieve formatted data from an Microsoft Excel spreadsheet and parsed using the MBS_SQL_Parse_Data series of Helper Functions.

The parameter list for this call is:

```
out text OUT_Text;
```

An example script is:

```
local text l_text;

call with name "MBS_Copy_From_Clipboard" in dictionary 5261, l_text;
```

## MBS_Show_Desktop_Alert

Use this call to display a desktop alert. It can be used instead of a warning statement to display information and does not require a mouse click to dismiss.

The parameter list for this call is:

```
in string IN_Title;
in text IN_Message;
in integer IN_Mode;
```

An example script is:

```
call with name "MBS_Show_Desktop_Alert" in dictionary 5261, "Title",
"Message", 1 {1 - Information, 2 - Warning, 3 - Error, 4 - Debug};
```

## MBS_Email_API

Use this Helper Function to send emails using the GP Power Tools email engine.

The parameter list for this call is:

```
in string IN_EmailFrom;
in string IN_EmailTo;
in string IN_EmailCC;
in string IN_EmailBCC;
in string IN_EmailSubject;
in text IN_EmailBody;
in text IN_EmailSignature;
in boolean IN_EmailSignatureDefault;
in text IN_EmailAttachments;
in boolean IN_EmailPreview;
in boolean IN_EmailAutoSend;
```

An example script is:

```
local string l_EmailFrom;
local string l_EmailTo;
local string l_EmailCC;
local string l_EmailBCC;
local string l_EmailSubject;
local text l_EmailBody;
local text l_EmailSignature;
local boolean l_EmailSignatureDefault;
local text l_EmailAttachments;
local boolean l_EmailPreview;
local boolean l_EmailAutoSend;

l_EmailTo = "email@domain.com";
l_EmailSubject = "Email API Test";
l_EmailBody = "This is a test of the Email API"+char(13);
l_EmailSignatureDefault = true;
l_EmailAttachments = l_EmailAttachments + "C:\Dex1000\Data\Dex.ini"+char(13);
l_EmailAttachments = l_EmailAttachments + "C:\Dex1100\Data\Dex.ini"+char(13);
l_EmailPreview = false;
l_EmailAutoSend = false;

call with name "MBS_Email_API" in dictionary 5261,
             l_EmailFrom, l_EmailTo, l_EmailCC, l_EmailBCC, l_EmailSubject,
             l_EmailBody, l_EmailSignature, l_EmailSignatureDefault,
             l_EmailAttachments, l_EmailPreview, l_EmailAutoSend;
```

## MBS_Add_Virtual_Field

This call is used to add a Virtual Field to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;

call with name "MBS_Add_Virtual_Field" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference };
```

## MBS_Add_Virtual_FieldPrompt

This call is used to add a Virtual Field and prompt to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
in      integer         IN_PromptFieldDict;
in      string          IN_PromptField;
in      integer         IN_PromptWidth;
in      string          IN_PromptFieldLabel;
out     reference       OUT_PromptReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;
local reference MBS_VF_PromptReference;

call with name "MBS_Add_Virtual_FieldPrompt" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference },
   5261 { Prompt Dictionary }, "VF_Prompt_01" { Prompt Field increment as needed 01-20 },
   128 { Prompt Width }, "<Prompt>" { Prompt Label },
   MBS_VF_PromptReference { Prompt Reference };
```

## MBS_Add_Virtual_FieldFormat

This call is used to add a Virtual Field and format to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
in      integer         IN_FormatFieldDict;
in      string          IN_FormatField;
in      integer         IN_FormatValue;
out     reference       OUT_FormatReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;
local reference MBS_VF_FormatReference;

call with name "MBS_Add_Virtual_FieldFormat" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference },
   5261 { Format Dictionary }, "VF_Format_Integer_01" {Format Field increment as needed 01-20},
   1 { Format Value },
   MBS_VF_FormatReference { Format Reference };
```

## MBS_Add_Virtual_FieldPromptLookup

This call is used to add a Virtual Field, prompt and lookup to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
in      integer         IN_PromptFieldDict;
in      string          IN_PromptField;
in      integer         IN_PromptWidth;
in      string          IN_PromptFieldLabel;
out     reference       OUT_PromptReference;
in      integer         IN_LookupFieldDict;
in      string          IN_LookupField;
out     reference       OUT_LookupReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;
local reference MBS_VF_PromptReference;
local reference MBS_VF_LookupReference;

call with name "MBS_Add_Virtual_FieldPromptLookup" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference },
   5261 { Prompt Dictionary }, "VF_Prompt_01" { Prompt Field increment as needed 01-20 },
   128 { Prompt Width }, "<Prompt>" { Prompt Label },
   MBS_VF_PromptReference { Prompt Reference },
   5261 { Lookup Dictionary }, "VF_Lookup_Button_01" {Lookup Field increment as needed 01-20},
   MBS_VF_LookupReference { Lookup Reference };
```

## MBS_Add_Virtual_FieldPromptFormat

This call is used to add a Virtual Field, prompt and format to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
in      integer         IN_PromptFieldDict;
in      string          IN_PromptField;
in      integer         IN_PromptWidth;
in      string          IN_PromptFieldLabel;
out     reference       OUT_PromptReference;
in      integer         IN_FormatFieldDict;
in      string          IN_FormatField;
in      integer         IN_FormatValue;
out     reference       OUT_FormatReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;
local reference MBS_VF_PromptReference;
local reference MBS_VF_FormatReference;

call with name "MBS_Add_Virtual_FieldPromptFormat" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference },
   5261 { Prompt Dictionary }, "VF_Prompt_01" { Prompt Field increment as needed 01-20 },
   128 { Prompt Width }, "<Prompt>" { Prompt Label },
   MBS_VF_PromptReference { Prompt Reference },
   5261 { Format Dictionary }, "VF_Format_Integer_01" {Format Field increment as needed 01-20},
   1 { Format Value },
   MBS_VF_FormatReference { Format Reference };
```

## MBS_Add_Virtual_FieldAll

This call is used to add a Virtual Field, prompt, lookup and format to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Left;
in      integer         IN_Height;
in      integer         IN_Width;
in      integer         IN_Vertical_Resize;
in      integer         IN_Horizontal_Resize;
in      boolean         IN_Editable;
in      boolean         IN_Border;
in      boolean         IN_Triggers;
in      string          IN_Default;
out     reference       OUT_FieldReference;
in      integer         IN_PromptFieldDict;
in      string          IN_PromptField;
in      integer         IN_PromptWidth;
in      string          IN_PromptFieldLabel;
out     reference       OUT_PromptReference;
in      integer         IN_LookupFieldDict;
in      string          IN_LookupField;
out     reference       OUT_LookupReference;
in      integer         IN_FormatFieldDict;
in      string          IN_FormatField;
in      integer         IN_FormatValue;
out     reference       OUT_FormatReference;
```

An example script is:

```
local reference MBS_VF_FieldReference;
local reference MBS_VF_PromptReference;
local reference MBS_VF_LookupReference;
local reference MBS_VF_FormatReference;

call with name "MBS_Add_Virtual_FieldAll" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   0 { Field Dictionary }, "<Field>" { Field },
   34 { Top }, 136 { Left }, 20  { Height }, 177 { Width },
   1 { Vertical Resize (T,C,B,TG,G,BG) }, 1 { Horizontal Resize (L,C,R,LG,G,RG) },
   true {Editable }, true { Border }, true { Create Triggers }, "" { Default Value/Caption },
   MBS_VF_FieldReference { Field Reference },
   5261 { Prompt Dictionary }, "VF_Prompt_01" { Prompt Field increment as needed 01-20 },
   128 { Prompt Width }, "<Prompt>" { Prompt Label },
   MBS_VF_PromptReference { Prompt Reference },
   5261 { Lookup Dictionary }, "VF_Lookup_Button_01" {Lookup Field increment as needed 01-20},
   MBS_VF_LookupReference { Lookup Reference },
   5261 { Format Dictionary }, "VF_Format_Integer_01" {Format Field increment as needed 01-20},
   1 { Format Value },
   MBS_VF_FormatReference { Format Reference };
```

## MBS_Add_Virtual_FieldLine

This call is used to add a Virtual Field line to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_FieldDict;
in      string          IN_Field;
in      integer         IN_Top;
in      integer         IN_Vertical_Resize;
out     reference       OUT_FieldReference;
```

An example script is:

```
local reference MBS_VF_LineReference;

call with name "MBS_Add_Virtual_FieldLine" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   5261 { Line Dictionary }, "VF_Line_01" { Line Field increment as needed 01-10 },
   34 { Top },
   3 { Vertical Resize (T,C,B) },
   MBS_VF_LineReference { Line Reference };
```

## MBS_Expand_Virtual_Field_Window

This call is used to expand a window and move existing fields down in prepation to add a Virtual Field to a window. It must be called from a Form Pre, Before Original trigger.

The parameter list for this call is:

```
in      integer         IN_Dict;
in      string          IN_Form;
in      string          IN_Window;
in      integer         IN_Vertical_Space;
in      boolean         IN_Navigation;
in      boolean         IN_Add_Line;
in      string          IN_SortBy;
in      string          IN_FieldList;
out     integer         OUT_Vertical_Pos;
```

An example script is:

```
local integer MBS_VF_VerticalPos;

call with name "MBS_Expand_Virtual_Field_Window" in dictionary 5261,
   0 { Form Dictionary }, "<Form>" { Form }, "<Window>" { Window },
   1*19 { Space to add }, true { Navigation Buttons }, true { Add Line },
   "Sort By" { Sort By Field }, "Note Absent Button - Toolbar;Note Present Button -
Toolbar;WindowHelp" { Additional Field List },
   MBS_VF_VerticalPos;
```

## MBS_Get_Field_Reference

This call is used to get a reference to an existing field or to a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Dict;
in string IN_Form;
in string IN_Window;
in integer IN_FieldDict;
in string IN_Field;
out reference OUT_Reference;
```

An example script is:

```
local reference l_reference;
call with name "MBS_Get_Field_Reference" in dictionary 5261,
Dictionary, "Form", "Window", FieldDictionary, "Field", l_reference;
if not empty(l_reference) then
    warning str(field(l_reference));
end if;
```

## MBS_Get_Virtual_Field

This call is used to get the value of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
inout anonymous field OUT_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Get_Virtual_Field" in dictionary 5261,
Dictionary, <Reference>, <Field>, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Virtual_Field

This call is used to set the value of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
inout anonymous field IN_Field_Value;
in boolean IN_Run_Flag;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Virtual_Field" in dictionary 5261,
Dictionary, <Reference>, <Field>, true {run script}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Map_Virtual_Field

This call is used to map the value of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
inout anonymous field IN_Field_Value;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Map_Virtual_Field" in dictionary 5261,
Dictionary, <Reference>, <Field>, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Virtual_Field_Caption

This call is used to get the caption of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
out string OUT_Field_Caption;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_caption;
call with name "MBS_Get_Virtual_Field_Caption" in dictionary 5261,
Dictionary, <Reference>, l_caption, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Virtual_Field_Caption

This call is used to set the caption of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
in string IN_Field_Caption;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Virtual_Field_Caption" in dictionary 5261,
Dictionary, <Reference>, "Caption", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Get_Virtual_Field_Tooltip

This call is used to get the tooltip of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
out string OUT_Field_Tooltip;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
local string l_tooltip;
call with name "MBS_Get_Virtual_Field_Tooltip" in dictionary 5261,
Dictionary, <Reference>, l_tooltip, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Set_Virtual_Field_Tooltip

This call is used to set the tooltip of a Virtual Field on a window.

The parameter list for this call is:

```
in integer IN_Prod_ID;
inout reference INOUT_Reference;
in string IN_Field_Tooltip;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Set_Virtual_Field_Tooltip" in dictionary 5261,
Dictionary, <Reference>, "Tooltip", l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

## MBS_Ask_Password

This call is used to ask for a Form Control Password .

The parameter list for this call is:

```
in string IN_PasswordID;
out boolean OUT_Success;
out integer OUT_Attempts;
```

An example script is:

```
local boolean l_success;
local integer l_attempts;
call with name "MBS_Ask_Password" in dictionary 5261,
"<PasswordID>", l_success, l_attempts;
```

## MBS_Control_Start

This call is used to start a Form Control ID for specified form in a dictionary.

The parameter list for this call is:

```
in string IN_ControlID;
in integer IN_DictID;
in string IN_Form;
in boolean IN_Modified;
```

An example script is:

```
call with name "MBS_Control_Start" in dictionary 5261, "XXXX", 0
{Dict}, "<Form Name>", false {modifiied};
```

## MBS_Control_Stop

This call is used to stop a Form Control ID for specified form in a dictionary.

The parameter list for this call is:

```
in string IN_ControlID;
in integer IN_DictID;
in string IN_Form;
```

An example script is:

```
call with name "MBS_Control_Stop" in dictionary 5261, "XXXX", 0
{Dict}, "<Form Name>";
```

## MBS_Control_Stop_All

This call is used to stop a Form Control ID for all forms.

The parameter list for this call is:

```
in string IN_ControlID;
```

An example script is:

```
call with name "MBS_Control_Stop_All" in dictionary 5261, "XXXX";
```

## MBS_Control_Update_Dialog

This call is used to programmatically change the Warning used for the currently active Form Control ID and Rule.

The parameter list for this call is:

```
in string IN_Warning;
in integer IN_Mode; { 1 = info, 2 = warning, 3 = error, 4 = debug }
```

An example script is:

```
call with name "MBS_Control_Update_Dialog" in dictionary 5261,
"Message", Mode {1 = info, 2 = warning, 3 = error, 4 = debug};
```

## MBS_Control_Update_Expression

This call is used to programmatically change the Expression used for the currently active Form Control ID and Rule.

The parameter list for this call is:

```
in string IN_Expression;
```

An example script is:

```
call with name "MBS_Control_Update_Expression" in dictionary 5261,
"Expression";
```

## MBS_Get_First_Window

This call is used to get the first or main window on a form. It can be used with Form Control conditional scripts when needing to work with fields on the main window rather than the current window.

The parameter list for this call is:

```
in integer IN_Dict;
in string IN_Form;
out string OUT_Window;
```

An example script is:

```
local string l_window;
call with name "MBS_Get_First_Window", 0 { Dictionary } , "<Form>" {
Form }, l_window;
```

## MBS_Check_Resource_Exists

This call is used to check if a form, window or field resources actually exists. It can be used with Form Control conditional scripts when needing to work with fields other than than the current field.

The parameter list for this call is:

```
in integer IN_Dict;
in string IN_Form;
in string IN_Window;
in string IN_Field;
in boolean IN_Modified;
out integer OUT_Status;
```

An example script is:

```
local integer l_status;
call with name "MBS_Check_Resource_Exists" in dictionary 5261, 0
{Dict}, "<Form>", "<Window>", "<Field>", false {Modified}, l_status;
if l_status <> OKAY then
    warning str(l_status);
end if;
```

# Chapter 10: RW Functions

GP Power Tools has support for six generic Report Writer functions in the core Dynamics.dic dictionary which can be used in the Report Writer with any report as a user defined function in a calculated field.

To use the functions with GP Power Tools, the first two parameters passed in for each of the functions will be the Dictionary ID for GP Power Tools (5261) and the Script ID of a Runtime Execute Setup script.

The Dexterity sanScript code contained in the Script ID will then be executed allowing for the development of custom Report Writer functions. Use the Select Custom Script Purpose option on the Runtime Execute Setup window to automatically add the template code to handle the parameter passing into and out of the script using the MBS_Param_Get and MBS_Param_Set helper functions.

Below are the details of the RW Functions available from the system series:

- *rw_ReportStart*
- *rw_ReportEnd*
- *rw_TableHeaderString (Supports additional functions)*
- *rw_TableHeaderCurrency*
- *rw_TableLineString*
- *rw_TableLineCurrency*

- *rw_ReportStart Old Method*
- *rw_ReportEnd Old Method*
- *rw_TableHeaderString Old Method*
- *rw_TableHeaderCurrency Old Method*
- *rw_TableLineString Old Method*
- *rw_TableLineCurrency Old Method*

- RW_GetUserMasterAdditionalData (GP Power Tools)
- RW_GetUserMasterAdditionalPrompts (GP Power Tools)

You can use these functions to capture information off a report and store it in the log using the MBS_Auto_Log helper function. For example: you can capture the values of legends, calculated fields, report fields or values from any of the attached tables.

*While the Report Writer Functions were designed to work with the report start and end events and with a header and line type document such as seen in Sales Order Processing, you can use the functions and parameters as desired to achieve the results required.*

More detail on these functions is available from Knowledge Base (KB) article 888884:

http://support.microsoft.com/kb/888884

## rw_ReportStart

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_ReportStart  5261  "Script ID"  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_Status;

MBS_Status = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_String" in dictionary 5261,
"RW_RS_ReportStart", MBS_Status;
```

## rw_ReportEnd

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_ReportEnd  5261  "Script ID"  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_Status;

MBS_Status = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_String" in dictionary 5261,
"RW_RE_ReportEnd", MBS_Status;
```

## rw_TableHeaderString

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in integer iControl; {which piece of data to return}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_TableHeaderString  5261  "Script ID"
SOP_HDR_WORK.SOP Number  SOP_HDR_WORK.SOP Type  1  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_TableHeaderString;
local string MBS_Number;
local integer MBS_Type;
local integer MBS_Control;

call with name "MBS_Memory_Get_String" in dictionary 5261,
"RW_THS_Number", MBS_Number;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_THS_Type", MBS_Type;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_THS_Control", MBS_Control;
MBS_TableHeaderString = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_String" in dictionary 5261,
"RW_THS_TableHeaderString", MBS_TableHeaderString;
```

*The rw_TableHeaderString function supports additional methods for the RW_GetUserMasterAdditionalData and RW_GetUserMasterAdditionalPrompts report writer functions using a dictionary value of -5261 and the IDs of "User" and "UserPrompt" respectively.*

## rw_TableHeaderCurrency

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `currency` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in integer iControl; {which piece of data to return}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of currency is:

```
FUNCTION_SCRIPT(  rw_TableHeaderCurrency  5261  "Script ID"
SOP_HDR_WORK.SOP Number  SOP_HDR_WORK.SOP Type  1  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local currency MBS_TableHeaderCurrency;
local string MBS_Number;
local integer MBS_Type;
local integer MBS_Control;

call with name "MBS_Memory_Get_String" in dictionary 5261,
"RW_THC_Number", MBS_Number;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_THC_Type", MBS_Type;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_THC_Control", MBS_Control;
MBS_TableHeaderCurrency = 0.0000;

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_Currency" in dictionary 5261,
"RW_THC_TableHeaderCurrency", MBS_TableHeaderCurrency;
```

## rw_TableLineString

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in currency cSequenceOne; {control field 3}
in currency cSequenceTwo; {control field 4}
in integer iControl; {which piece of data to return}
```

*To use the rw_TableLineString report writer function we need to be able to pass the two sequence fields as currency data type. So to use the Sales Order Processing fields SOP_LINE_WORK.Line Item Sequence and SOP_LINE_WORK.Component Sequence, we will need to create two calculated fields to convert them from a long integer to a currency data type.*

Calculated Field **(C) Line Item Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Line Item Sequence * 1.00000*.

Calculated Field **(C) Component Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Component Sequence * 1.00000*.

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT( rw_TableLineString 5261 "Script ID"
SOP_LINE_WORK.SOP Number SOP_LINE_WORK.SOP Type (C) Line Item
Sequence (C) Component Sequence 1 )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_TableLineString;
local string MBS_Number;
local integer MBS_Type;
local currency MBS_SequenceOne;
local currency MBS_SequenceTwo;
local integer MBS_Control;

call with name "MBS_Memory_Get_String" in dictionary 5261,
"RW_TLS_Number", MBS_Number;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_TLS_Type", MBS_Type;
call with name "MBS_Memory_Get_Currency" in dictionary 5261,
"RW_TLS_SequenceOne", MBS_SequenceOne;
call with name "MBS_Memory_Get_Currency" in dictionary 5261,
"RW_TLS_SequenceTwo", MBS_SequenceTwo;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_TLS_Control", MBS_Control;
MBS_TableLineString = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_String" in dictionary 5261,
"RW_TLS_TableLineString", MBS_TableLineString;
```

## rw_TableLineCurrency

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `currency` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in currency cSequenceOne; {control field 3}
in currency cSequenceTwo; {control field 4}
in integer iControl; {which piece of data to return}
```

*To use the rw_TableLineCurrency report writer function we need to be able to pass the two sequence fields as currency data type. So to use the Sales Order Processing fields SOP_LINE_WORK.Line Item Sequence and SOP_LINE_WORK.Component Sequence, we will need to create two calculated fields to convert them from a long integer to a currency data type.*

Calculated Field **(C) Line Item Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Line Item Sequence * 1.00000*.

Calculated Field **(C) Component Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Component Sequence * 1.00000*.

An example of how it would be called from the Report Writer for a calculated field with a Result Type of currency is:

```
FUNCTION_SCRIPT( rw_TableLineCurrency 5261 "Script ID"
SOP_LINE_WORK.SOP Number SOP_LINE_WORK.SOP Type (C) Line Item
Sequence (C) Component Sequence 1 )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local currency MBS_TableLineCurrency;
local string MBS_Number;
local integer MBS_Type;
local currency MBS_SequenceOne;
local currency MBS_SequenceTwo;
local integer MBS_Control;

call with name "MBS_Memory_Get_String" in dictionary 5261,
"RW_TLC_Number", MBS_Number;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_TLC_Type", MBS_Type;
call with name "MBS_Memory_Get_Currency" in dictionary 5261,
"RW_TLC_SequenceOne", MBS_SequenceOne;
call with name "MBS_Memory_Get_Currency" in dictionary 5261,
"RW_TLC_SequenceTwo", MBS_SequenceTwo;
call with name "MBS_Memory_Get_Long" in dictionary 5261,
"RW_TLC_Control", MBS_Control;
MBS_TableLineCurrency = 0.00000;

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Memory_Set_Currency" in dictionary 5261,
"RW_TLC_TableLineCurrency", MBS_TableLineCurrency;
```

## rw_ReportStart Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_ReportStart  5261  "Script ID"  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_Status;

MBS_Status = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261, "ReportStart",
MBS_Status;
```

## rw_ReportEnd Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_ReportEnd  5261  "Script ID"  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_Status;

MBS_Status = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261, "ReportEnd",
MBS_Status;
```

## rw_TableHeaderString Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in integer iControl; {which piece of data to return}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_TableHeaderString  5261  "Script ID"
SOP_HDR_WORK.SOP Number  SOP_HDR_WORK.SOP Type  1  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_TableHeaderString;
local string MBS_Number;
local integer MBS_Type;
local integer MBS_Control;
local string MBS_String;

call with name "MBS_Param_Get" in dictionary 5261, "Number",
MBS_Number;
call with name "MBS_Param_Get" in dictionary 5261, "Type",
MBS_String;
MBS_Type = integer(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "Control",
MBS_String;
MBS_Control = integer(value(MBS_String));
MBS_TableHeaderString = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261,
"TableHeaderString", MBS_TableHeaderString;
```

## rw_TableHeaderCurrency Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `currency` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in integer iControl; {which piece of data to return}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of currency is:

```
FUNCTION_SCRIPT(  rw_TableHeaderCurrency  5261  "Script ID"
SOP_HDR_WORK.SOP Number  SOP_HDR_WORK.SOP Type  1  )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local currency MBS_TableHeaderCurrency;
local string MBS_Number;
local integer MBS_Type;
local integer MBS_Control;
local string MBS_String;

call with name "MBS_Param_Get" in dictionary 5261, "Number",
MBS_Number;
call with name "MBS_Param_Get" in dictionary 5261, "Type",
MBS_String;
MBS_Type = integer(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "Control",
MBS_String;
MBS_Control = integer(value(MBS_String));
MBS_TableHeaderCurrency = 0.0000;

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261,
"TableHeaderCurrency", str(MBS_TableHeaderCurrency);
```

## rw_TableLineString Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in currency cSequenceOne; {control field 3}
in currency cSequenceTwo; {control field 4}
in integer iControl; {which piece of data to return}
```

*To use the rw_TableLineString report writer function we need to be able to pass the two sequence fields as currency data type. So to use the Sales Order Processing fields SOP_LINE_WORK.Line Item Sequence and SOP_LINE_WORK.Component Sequence, we will need to create two calculated fields to convert them from a long integer to a currency data type.*

Calculated Field **(C) Line Item Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Line Item Sequence * 1.00000*.

Calculated Field **(C) Component Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Component Sequence * 1.00000*.

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT( rw_TableLineString 5261 "Script ID"
SOP_LINE_WORK.SOP Number SOP_LINE_WORK.SOP Type (C) Line Item
Sequence (C) Component Sequence 1 )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local string MBS_TableLineString;
local string MBS_Number;
local integer MBS_Type;
local currency MBS_SequenceOne;
local currency MBS_SequenceTwo;
local integer MBS_Control;
local string MBS_String;

call with name "MBS_Param_Get" in dictionary 5261, "Number",
MBS_Number;
call with name "MBS_Param_Get" in dictionary 5261, "Type",
MBS_String;
MBS_Type = integer(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "SequenceOne",
MBS_String;
MBS_SequenceOne = currency(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "SequenceTwo",
MBS_String;
MBS_SequenceTwo = currency(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "Control",
MBS_String;
MBS_Control = integer(value(MBS_String));
MBS_TableLineString = "";

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261,
"TableLineString", MBS_TableLineString;
```

## rw_TableLineCurrency Old Method

This report writer function can be used in the Report Writer as a user defined function in a calculated field. The first two parameters passed in need to be the Dictionary ID for GP Power Tools (5261) and the Script ID of the Runtime Execute Setup script to be executed.

The returned value for this report writer function is a `currency` and the input parameter list for this report writer function is:

```
in integer dict_id; {Dictionary ID}
in string script_id; {Script ID}
in string sNumber; {control field 1}
in integer sType; {control field 2}
in currency cSequenceOne; {control field 3}
in currency cSequenceTwo; {control field 4}
in integer iControl; {which piece of data to return}
```

*To use the rw_TableLineCurrency report writer function we need to be able to pass the two sequence fields as currency data type. So to use the Sales Order Processing fields SOP_LINE_WORK.Line Item Sequence and SOP_LINE_WORK.Component Sequence, we will need to create two calculated fields to convert them from a long integer to a currency data type.*

Calculated Field **(C) Line Item Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Line Item Sequence * 1.00000*.

Calculated Field **(C) Component Sequence** is defined as result type currency with the expression of *SOP_LINE_WORK.Component Sequence * 1.00000*.

An example of how it would be called from the Report Writer for a calculated field with a Result Type of currency is:

```
FUNCTION_SCRIPT( rw_TableLineCurrency 5261 "Script ID"
SOP_LINE_WORK.SOP Number SOP_LINE_WORK.SOP Type (C) Line Item
Sequence (C) Component Sequence 1 )
```

The template Runtime Execute Setup script added by the Helper Function Assistant window is:

```
local currency MBS_TableLineCurrency;
local string MBS_Number;
local integer MBS_Type;
local currency MBS_SequenceOne;
local currency MBS_SequenceTwo;
local integer MBS_Control;
local string MBS_String;

call with name "MBS_Param_Get" in dictionary 5261, "Number",
MBS_Number;
call with name "MBS_Param_Get" in dictionary 5261, "Type",
MBS_String;
MBS_Type = integer(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "SequenceOne",
MBS_String;
MBS_SequenceOne = currency(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "SequenceTwo",
MBS_String;
MBS_SequenceTwo = currency(value(MBS_String));
call with name "MBS_Param_Get" in dictionary 5261, "Control",
MBS_String;
MBS_Control = integer(value(MBS_String));
MBS_TableLineCurrency = 0.00000;

{ Add your code below here }

{ Add your code above here }

call with name "MBS_Param_Set" in dictionary 5261,
"TableLineCurrency", str(MBS_TableLineCurrency);
```

## RW_GetUserMasterAdditionalData

This report writer function can be used in the Report Writer as a user defined function in a calculated field to retrieve data stored in the User Setup Additional Information window.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in 'User ID' IN_UserID; {User ID variable}
in integer IN_Position; {which field to return: Field 1 to 20}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  RW_GetUserMasterAdditionalData  SY_Users_MSTR.User
ID  1  )
```

However, as this report writer function is in the GP Power Tools dictionary, it cannot be used from reports in other dictionaries. To get around this limitation the functionality has also been added to the rw_TableHeaderString report writer function using a negative Dictionary ID.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Negative Dictionary ID: -5261}
in string IN_Function; {String constant: "User"}
in string IN_UserID; {User ID variable}
in integer IN_NotUsed; {Integer constant: 0}
in integer IN_Position; {which field to return: Field 1 to 20}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_TableHeaderString  -5261  "User"
SY_Users_MSTR.User ID  0  1  )
```

## RW_GetUserMasterAdditionalPrompts

This report writer function can be used in the Report Writer as a user defined function in a calculated field to retrieve prompts used for the User defined fields in the User Setup Additional Information window.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer IN_Position; {which field to return: Field 1 to 6}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  RW_GetUserMasterAdditionalPrompts  1  )
```

However, as this report writer function is in the GP Power Tools dictionary, it cannot be used from reports in other dictionaries. To get around this limitation the functionality has also been added to the rw_TableHeaderString report writer function using a negative Dictionary ID.

The returned value for this report writer function is a `string` and the input parameter list for this report writer function is:

```
in integer dict_id; {Negative Dictionary ID: -5261}
in string IN_Function; {String constant: "UserPrompt"}
in string IN_UserID; {String constant: ""}
in integer IN_NotUsed; {Integer constant: 0}
in integer IN_Position; {which field to return: Field 1 to 6}
```

An example of how it would be called from the Report Writer for a calculated field with a Result Type of string is:

```
FUNCTION_SCRIPT(  rw_TableHeaderString  -5261  "UserPrompt"  ""  0
1  )
```

# Chapter 11: Service Procedures

GP Power Tools has five Service Procedures which can be called to perform custom actions within Microsoft Dynamics GP. These Service Procedures can be called from Dexterity, Visual Studio (Visual C# or Visual Basic.Net) or from Web Services.

To use the Service Procedures with GP Power Tools, the first parameter passed in will be the Script ID of a Runtime Execute Setup script.

The Dexterity sanScript code contained in the Script ID will then be executed allowing for the development of custom Service Procedures. Use the Select Custom Script Purpose option on the Runtime Execute Setup window to automatically add the template code to handle the parameter passing into and out of the script using listbox fields or collections of strings.

*Due to limitations in Dexterity, the maximum size of data passed into and out of GP Power Tools Service Procedures is limited to the maximum size of a text variable being 32K (32767 bytes).*

Below are the details of the Service Procedures system series:

- *ServiceCreateCustom*
- *ServiceDeleteCustom*
- *ServiceGetCustom*
- *ServiceUpdateCustom*
- *ServicePostCustom*

## ServiceCreateCustom

This service procedure can be used to make custom "Create" code which can be called as a web service or from Visual Studio or Dexterity. The first parameter passed in needs to be the Script ID of the Runtime Execute Setup script to be executed.

### Web Service

Details for calling the service procedure as a web service are:

| | | | |
|---|---|---|---|
| Name: | ServiceCreateCustom | | |
| URI Template: | /Custom/Create({ScriptID}) | | |
| Header Value: | GP-Custom-Action=Post | | |
| Request Type: | Custom (POST) | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| Payload: | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| Returns: | Status | (Short) | Returned Status Code |

URL Example:

https://domain.com/gpservice/Tenants(DefaultTenant)/Companies(Fabrikam,%20Inc.)/GPPowerTools/ Custom/Create({ScriptID})

### Visual Studio Call

Details for calling the service procedure using Visual Studio are:

| | | | |
|---|---|---|---|
| Name: | ServiceCreateCustom | | |
| Qualified Name: | GPPowerTools.Procedures.ServiceCreateCustom | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| | Status | (out Short) | Returned Status Code |

Invoke Example:

Application.GPPowerTools.Procedures.ServiceCreateCustom.Invoke(ScriptID, InStringList, Status;)

## Dexterity Call

Details for calling the service procedure using Dexterity are:

Name: ServiceCreateCustom

Parameters:

```
using System.Collections;
using System.Collections.Generic;

in      string                  ScriptID;
in      List<System.String>     InStringList;
out     integer                 Status;
```

Call Example:

```
call with name "ServiceCreateCustom" in dictionary 5261, ScriptID, InStringList, Status;
```

## ServiceDeleteCustom

This service procedure can be used to make custom "Delete" code which can be called as a web service or from Visual Studio or Dexterity. The first parameter passed in needs to be the Script ID of the Runtime Execute Setup script to be executed.

## Web Service

Details for calling the service procedure as a web service are:

| | | | |
|---|---|---|---|
| Name: | ServiceDeleteCustom | | |
| URI Template: | /Custom/Delete({ScriptID}) | | |
| Header Value: | GP-Custom-Action=Post | | |
| Request Type: | Custom (POST) | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| Payload: | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| Returns: | Status | (Short) | Returned Status Code |

URL Example:

https://domain.com/gpservice/Tenants(DefaultTenant)/Companies(Fabrikam,%20Inc.)/GPPowerTools/ Custom/Delete({ScriptID})

## Visual Studio Call

Details for calling the service procedure using Visual Studio are:

| | | | |
|---|---|---|---|
| Name: | ServiceDeleteCustom | | |
| Qualified Name: | GPPowerTools.Procedures.ServiceDeleteCustom | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| | Status | (out Short) | Returned Status Code |

Invoke Example:

Application.GPPowerTools.Procedures.ServiceDeleteCustom.Invoke(ScriptID, InStringList, Status;)

## Dexterity Call

Details for calling the service procedure using Dexterity are:

Name: ServiceDeleteCustom

Parameters:

```
using System.Collections;
using System.Collections.Generic;

in      string                  ScriptID;
in      List<System.String>     InStringList;
out     integer                 Status;
```

Call Example:

```
call with name "ServiceDeleteCustom" in dictionary 5261, ScriptID, InStringList, Status;
```

## ServiceGetCustom

This service procedure can be used to make custom "Get" code which can be called as a web service or from Visual Studio or Dexterity. The first parameter passed in needs to be the Script ID of the Runtime Execute Setup script to be executed.

### Web Service

Details for calling the service procedure as a web service are:

Name: ServiceGetCustom

URI Template: /Custom/Get({ScriptID})

Header Value: GP-Custom-Action=Post

Request Type: Custom (POST)

| | | | |
|---|---|---|---|
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| Payload: | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| Returns: | List<OutStringList> | (List<String>) | Passed out data as collection of strings |
| | Status | (Short) | Returned Status Code |

URL Example:

https://domain.com/gpservice/Tenants(DefaultTenant)/Companies(Fabrikam,%20Inc.)/GPPowerTools/ Custom/Get({ScriptID})

### Visual Studio Call

Details for calling the service procedure using Visual Studio are:

Name: ServiceGetCustom

Qualified Name: GPPowerTools.Procedures.ServiceGetCustom

| | | | |
|---|---|---|---|
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| | List<OutStringList> | (List<String>) | Passed out data as collection of strings |
| | Status | (out Short) | Returned Status Code |

Invoke Example:

Application.GPPowerTools.Procedures.ServiceGetCustom.Invoke(ScriptID, InStringList, OutStringList, Status;)

## Dexterity Call

Details for calling the service procedure using Dexterity are:

Name: ServiceGetCustom

Parameters:

```
using System.Collections;
using System.Collections.Generic;

in      string                  ScriptID;
in      List<System.String>     InStringList;
out     List<System.String>     OutStringList;
out     integer                 Status;
```

Call Example:

```
call with name "ServiceGetCustom" in dictionary 5261, ScriptID, InStringList,
OutStringList, Status;
```

## ServiceUpdateCustom

This service procedure can be used to make custom "Update" code which can be called as a web service or from Visual Studio or Dexterity. The first parameter passed in needs to be the Script ID of the Runtime Execute Setup script to be executed.

### Web Service

Details for calling the service procedure as a web service are:

| | | | |
|---|---|---|---|
| Name: | ServiceUpdateCustom | | |
| URI Template: | /Custom/Update({ScriptID}) | | |
| Header Value: | GP-Custom-Action=Post | | |
| Request Type: | Custom (POST) | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| Payload: | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| Returns: | Status | (Short) | Returned Status Code |

URL Example:

https://domain.com/gpservice/Tenants(DefaultTenant)/Companies(Fabrikam,%20Inc.)/GPPowerTools/ Custom/Update({ScriptID})

### Visual Studio Call

Details for calling the service procedure using Visual Studio are:

| | | | |
|---|---|---|---|
| Name: | ServiceUpdateCustom | | |
| Qualified Name: | GPPowerTools.Procedures.ServiceUpdateCustom | | |
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| | Status | (out Short) | Returned Status Code |

Invoke Example:

Application.GPPowerTools.Procedures.ServiceUpdateCustom.Invoke(ScriptID, InStringList, Status;)

## Dexterity Call

Details for calling the service procedure using Dexterity are:

Name: ServiceUpdateCustom

Parameters:

```
using System.Collections;
using System.Collections.Generic;

in      string                  ScriptID;
in      List<System.String>     InStringList;
out     integer                 Status;
```

Call Example:

```
call with name "ServiceUpdateCustom" in dictionary 5261, ScriptID, InStringList, Status;
```

## ServicePostCustom

This service procedure can be used to make custom "Post" code which can be called as a web service or from Visual Studio or Dexterity. The first parameter passed in needs to be the Script ID of the Runtime Execute Setup script to be executed.

### Web Service

Details for calling the service procedure as a web service are:

Name: ServicePostCustom

URI Template: /Custom/Post({ScriptID})

Header Value: GP-Custom-Action=Post

Request Type: Custom (POST)

| | | | |
|---|---|---|---|
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| Payload: | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| Returns: | List<OutStringList> | (List<String>) | Passed out data as collection of strings |
| | Status | (Short) | Returned Status Code |

URL Example:

https://domain.com/gpservice/Tenants(DefaultTenant)/Companies(Fabrikam,%20Inc.)/GPPowerTools/ Custom/Post({ScriptID})?GP-Custom-Action=Post

### Visual Studio Call

Details for calling the service procedure using Visual Studio are:

Name: ServicePostCustom

Qualified Name: GPPowerTools.Procedures.ServicePostCustom

| | | | |
|---|---|---|---|
| Parameters: | ScriptID | (String) | Runtime Execute Setup Script ID |
| | List<InStringList> | (List<String>) | Passed in data as collection of strings |
| | List<OutStringList> | (List<String>) | Passed out data as collection of strings |
| | Status | (out Short) | Returned Status Code |

Invoke Example:

Application.GPPowerTools.Procedures.ServicePostCustom.Invoke(ScriptID, InStringList, OutStringList, Status;)

## Dexterity Call

Details for calling the service procedure using Dexterity are:

Name: ServicePostCustom

Parameters:

```
using System.Collections;
using System.Collections.Generic;

in      string                  ScriptID;
in      List<System.String>     InStringList;
out     List<System.String>     OutStringList;
out     integer                 Status;
```

Call Example:

```
call with name "ServicePostCustom" in dictionary 5261, ScriptID, InStringList,
OutStringList, Status;
```

# Chapter 12: Developer APIs

GP Power Tools has several external APIs available for use by other developers.

Below are the details of the Developer APIs:

- *MBS_Email_API*
- *MBS_WindowPositionCheck*
- *MBS_WindowPositionMemory*
- *MBS_WindowPositionMemoryResize*
- *MBS_CompanyColorGet*
- *MBS_CompanyColorGetRGB*

## MBS_Email_API

This Developer API can be called from another Dexterity product to send emails using the GP Power Tools email engine.

The parameter list for this call is:

```
in string IN_EmailFrom;
in string IN_EmailTo;
in string IN_EmailCC;
in string IN_EmailBCC;
in string IN_EmailSubject;
in text IN_EmailBody;
in text IN_EmailSignature;
in boolean IN_EmailSignatureDefault;
in text IN_EmailAttachments;
in boolean IN_EmailPreview;
in boolean IN_EmailAutoSend;
```

An example script is:

```
local string l_EmailFrom;
local string l_EmailTo;
local string l_EmailCC;
local string l_EmailBCC;
local string l_EmailSubject;
local text l_EmailBody;
local text l_EmailSignature;
local boolean l_EmailSignatureDefault;
local text l_EmailAttachments;
local boolean l_EmailPreview;
local boolean l_EmailAutoSend;

l_EmailTo = "email@domain.com";
l_EmailSubject = "Email API Test";
l_EmailBody = "This is a test of the Email API"+char(13);
l_EmailSignatureDefault = true;
l_EmailAttachments = l_EmailAttachments + "C:\Dex1000\Data\Dex.ini"+char(13);
l_EmailAttachments = l_EmailAttachments + "C:\Dex1100\Data\Dex.ini"+char(13);
l_EmailPreview = false;
l_EmailAutoSend = false;

call with name "MBS_Email_API" in dictionary 5261,
             l_EmailFrom, l_EmailTo, l_EmailCC, l_EmailBCC, l_EmailSubject,
             l_EmailBody, l_EmailSignature, l_EmailSignatureDefault,
             l_EmailAttachments, l_EmailPreview, l_EmailAutoSend;
```

## MBS_WindowPositionCheck

This Developer API can be called from another Dexterity product to temporarily disable the Window Position Check which prevents windows from opening outside of the visible desktop.

The parameter list for this call is:

```
in boolean IN_Active;
```

An example script is:

```
call with name "MBS_WindowPositionCheck" in dictionary 5261,
    false;
```

Remember to re-enable the feature with a second call after your code has completed.

## MBS_WindowPositionMemory

This Developer API can be called from another Dexterity product to temporarily disable the Window Position Memory feature which can move windows to their previous location on the desktop.

The parameter list for this call is:

```
in boolean IN_Active;
```

An example script is:

```
call with name "MBS_WindowPositionMemory" in dictionary 5261,
    false;
```

Remember to re-enable the feature with a second call after your code has completed.

## MBS_WindowPositionMemoryResize

This Developer API can be called from another Dexterity product to temporarily disable the Window Position Memory Resizing feature which can resize windows to their previous size on the desktop.

The parameter list for this call is:

```
in boolean IN_Active;
```

An example script is:

```
call with name "MBS_WindowPositionMemoryResize" in dictionary 5261,
    false;
```

Remember to re-enable the feature with a second call after your code has completed.

## MBS_CompanyColorGet

This Developer API can be called from Visual Studio to obtain the RGB color settings as per the current Company Color Theme in use. The color settings can then be used to set the BackColor of Windows and Controls of WinForms so that they match the colors of the Dexterity windows.

The parameter list for this call is:

```
out long OUT_ApplicationBackground;
out long OUT_WindowBackground;
out long OUT_WindowToolbar;
out long OUT_ButtonBackground;
out long OUT_ScrollLineBackground;
```

An example C# script is:

```
Microsoft.Dexterity.Applications.GpPowerTools.Functions.MbsCompanyCo
lorGetRgb.Invoke(out ApplicationBackground, out WindowBackground,
out WindowToolbar, out FieldBackground,out ScrollLineBackground);
if (WindowBackground > 0) {
    form.BackColor = Color.FromArgb(WindowBackground);
}
```

Primarily the Window Background (Mode 2) and Field Background (Mode 4) can be used for Field colors. Pushbuttons should use (253, 253, 253) for their color to match Dexterity windows.

## MBS_CompanyColorGetRGB

This Developer API can be called from Visual Studio to obtain an individual RGB color setting from the current Company Color Theme in use. The color setting can then be used to set the BackColor of Windows and Controls of WinForms so that they match the colors of the Dexterity windows.

The parameter list for this call is:

```
in integer IN_Mode;
out integer OUT_red_level, OUT_green_level, OUT_blue_level;
```

An example C# script is:

```
Microsoft.Dexterity.Applications.GpPowerTools.Functions.MbsCompanyCo
lorGetRgb.Invoke(2, out redColor, out greenColor, out blueColor);
if ((redColor + greenColor + blueColor) > 0) {
    form.BackColor = Color.FromArgb(redColor, greenColor, blueColor);
}
```

Primarily the Window Background (Mode 2) and Field Background (Mode 4) can be used for Field colors. Pushbuttons should use (253, 253, 253) for their color to match Dexterity windows.

# GP Power Tools Index

## B



## C

## D

## E

## F

## G

## H

## I



## J



## K



## L

## M

## N



## O

## P

## Q



## R

## S

## T

## U

## V



## W

## X



** End of document - GPPTools.docx - DM - 4 March 2025 **