WINSHUTTLE™

Designing Faster Solutions with Winshuttle Composer

Summary

This white paper provides an overview of system optimizations designed to improve solutions created with Winshuttle Composer. It will also help you understand how the design of a solution impacts the solution's speed and performance. Tools for troubleshooting solutions identifying problem areas, and correcting/improving performance issues are also described.

Table of contents

Overview

Winshuttle uses SharePoint as the interface for creating forms and workflows, and many other ‘out-of-box’ features to help business users quickly create robust, Enterprise-level solutions with minimal, if any, code writing.

But the average user does not easily understand that designing and processing a Winshuttle form is not the same as designing a standalone website that users can search and interact with. It is important to help customers understand that Winshuttle applications don’t create experiences akin to shopping on popular Web sites.

As a developer, you are creating Enterprise form and workflow applications. But often the client organization doesn't have the internal infrastructure to support the best possible performance, and Winshuttle products do not use the same web technology used by mainstream internet sites.

This guide will help you understand, rate, troubleshoot, and improve solution performance.

Where do I begin?

Optimizing and troubleshooting a form can be overwhelming when the pressure is on. If the options above still don’t adequately narrow down portions of the solution that can directly relate to poor performance, dissect the solution one component at a time and measure the results using the tools described in the tools section (Blackbird, IE/Chrome developer tools, etc.).

When you begin troubleshooting or optimizing a form, save the solution under a different name and then deploy it. Viewing results after changes have been made can then be easily compared to the current solution.

Below are some general strategies for testing the components in a solution that can help you find areas for improvement. Each option should be completed one at time, after which the solution should be deployed and then tested. The options can be completed in any order.

1. Remove all data connections. If this drastically improves response time, review how each data connection is being used – as a drop down, lookup, default value, etc. – and determine if any of the other options above can assist with improving the speed of the data being pulled in from the data connections
2. Remove all customer JavaScript functions. If improves performance, restore the functions and remove one at a time to narrow down the problematic functions.
3. Delete all unused form elements, controls and views.

Remove all but the Originator and Process swimlanes. If this improves performance, determine if all swimlanes are necessary. For example, swimlanes that only contain Notification nodes could possibly be eliminated and the notification sent via the Send Email plug-in.

Defining peak performance

Solution performance is defined by multiple components. Some enterprises have data centers located in close proximity to each other connected by high-bandwidth fiber optic links. Other organizations have all servers in one data center.

Regardless of configuration, one thing must remain consistent: **connectivity to the Winshuttle environment.**

In general, **you should be able to go to any server in your environment and ping the other servers and have less than 1 millisecond response time** over a period of 10 minutes. Any higher response times are likely a contributing factor to issues within your SharePoint and Winshuttle environments' performance.

In addition, a good solution should be:

- **Reliable.** The solution works as designed (repeated results) at a high rate when available.
- **Stable**. The solution is highly available and rarely unavailable.
- **Functional**. The solution does what was designed to do.
- **Responsive**. The solution is fast and operates in a timely fashion.
- **Cost effective**. The solution meets the budgetary needs of the project.

Using the criteria above, rating solution performance could be summed up as follows:

- **Good performance**: All of the above criteria are met.
- **Fair performance**: Some (3-4) of the above criteria are met.
- **Poor performance**: 2 or fewer criteria are met.

Performance and **speed** are related, but they are not the same. Performance is the act of completing a task in a consistent manner and achieving the desired level of results. Speed defines how quickly/timely a task is performed.

Unfortunately, most of the time you simply can't 'have it all', nor can you control all the variables to allow your customer to have it all.

Architects should align available performance and functionality to the expectations of the solution. Questions about performance metrics and priorities are often overshadowed by management questions about how quickly a project can be completed and how much it will cost.

Many of the performance attributes are simply implied to be in the “*we-want-it-all*” area when we know that generally isn’t feasible. And when a solution is delivered that meets all the defined and agreed upon requirements, it is deemed too slow and project is momentarily derailed until performance and speed are improved.

This can be particularly frustrating because most of the time a requirement for speed is not defined, and the system and environment in question *(servers, network, desktops, etc.) aren’t tested* to generate a baseline against which the speed of a Winshuttle solution can be measured.

Good performance is relatively subjective, and can only be defined when performance baselines are determined. Speed and performance requirements should then be defined based upon baseline performance measurements.

The remaining sections of this document can help you find where the speed of the *system* may be lacking to determine if the issue is the fault of the solution design. In addition, troubleshooting options are provided that can help increase the speed of a form.

Tuning performance

As a Winshuttle developer, you must live within the confines of the system you are given, but the following tools and approaches will help you identify if that system is even capable of supporting the desired speed.

These tools will help you answer questions such as

- How much the amount of server memory impacts performance
- How long does it take from the time a form is launched until it is ready for use

Prepare the Baseline

In defining baseline performance you first need to define the baseline for the hardware for servers based on the number of users and the location of the servers. You must also measure the performance of a blank Winshuttle form and workflow without rules, data connections, and Winshuttle Transaction and Query scripts. This baseline represents the best performance that can be attained in your environment without changing any variables.

Minimum Hardware Requirements by User Count and Server Role

Below are the minimum hardware requirements based on Winshuttle and Microsoft best practices.

Less than 1000 Users

Server Role	CPUs	RAM	Additional Notes
SharePoint Web Front End (WFE)	64bit 4 Cores	12 GB	Each SharePoint WFE can support up on average 10,000 users
Database	64bit 4 Cores	8 GB	Supports 1000 users
Winshuttle Integration Server	64bit 4 Cores	8 GB	Can support 7 to 8 Query or Transaction Request a second

1000 to 10,000 Users

Server Role	CPUs	RAM	Additional Notes
SharePoint WFE #1	64bit 4 Cores	12 GB	Each SharePoint WFE can support up on average 10,000 users
SharePoint WFE #2	64bit 4 Cores	12 GB	Each SharePoint WFE can support up on average 10,000 users
Database	64 bit 8 Cores	16 GB	1000 to 10,000 users
Winshuttle Integration Server	64bit 4 Cores	8 GB	Can support 7 to 8 Query or Transaction Request a second

Evaluating performance: Know your tools

The following tools can help you establish a performance baseline for a solution in your environment:

- A blank Composer solution
- Ping utility
- Browser developer tools
- Blackbird

Blank Composer Solution

Whether or not you have the minimum hardware specified in the previous section, you need a baseline for your environment's performance.

Start by creating **a blank Composer solution** that has no rules, data connections, or Transaction/Query scripts. This may not be a realistic solution, but you can use it to get a baseline metric. It will also later help identify potential performance characteristics of your solution.

Two key performance metrics that should be obtained from this test are:

- **Time to open form**: The amount of time required to open the form.
- **Time to submit form**: The amount of time required to submit the form and return to the home page.

Each of these metrics reveals best-case performance for in the current environment given the hardware and network configuration. You can also experiment with different configurations and measure the effects on performance. For example, you could add RAM and/or CPUs to each server role (specifically the SharePoint WFE and Database servers) and then monitor one or both of the performance metrics listed above.

Once you have basic performance metrics you can add one solution component at a time (data connections, Transaction/Query scripts, rules, etc.), measure the impact on performance, and determine which design decisions or desired functionality may cause improve or degrade performance for Time to Open Form or Time to Submit Form.

Ping utility

In order to use the Ping tool you must be logged into your development/QA or production servers. In some organizations this is not permitted to business users or developers. (In these cases, ask your server or network administrators to perform this test.)

1. Press Windows Key+X
2. Click Command Prompt (Admin)
3. Type the following command:

```
ping [server name or IP address] /t
```

This command will continue pinging the server until you press CTRL+C.

Once the ping stops you will see the statistics for the ping.

Ideally you should see **ping times of 1ms or less for each server you ping** in your Winshuttle environment. Ping times greater than 1ms may indicate issues that need to be addressed and areas where performance can be improved.

Browser Developer Tools (Internet Explorer/Chrome)

IE and Chrome Developer tools enable you to identify issues in your Winshuttle Form that could be causing performance issues. Browser developer tools can help you to understand what happens when your form is loaded and then submitted.

(The Links and References section at the end of this paper includes links to pages describing these tools in greater detail.)

When using these tools to report what your form is doing and how quickly it is doing it, it is best to get the form in the state right at the point you want to run the test, and then quickly stop the reporting right after the desired action be tracked as finished.

For example, if you are trying to determine what is going on when a form is opened, first navigate to the location where the form is to be launched and then start the recording just before you open the form and then stop the recording after the form has loaded. This will help eliminate unnecessary noise from the results.

Common examples of what to look for when using browser developer tools:

- How long does it take to receive a response from the server after a request is sent?
- Once a response from the server is received, how long does it take to process it?
- How long and how many times is the form looping through JavaScript functions? If you have custom JavaScript in your solution and you see that function has a high number of times it is being called, perhaps it is getting stuck in a loop or the conditions of when it is being called are not specific enough to minimize the run count.

Blackbird

The Winshuttle **Composer Blackbird utility** is a powerful tool that can help you find performance issues in the Winshuttle application as well as bugs within your form.

To open Blackbird: Open any Winshuttle Composer form in a browser, and then press **CTRL+F12**.

In Composer versions 11.0.2 and prior, Blackbird does not display timestamps by default, but you can add them using the procedure below.

Adding Timestamps to Blackbird in Composer v11.0.2 and earlier:

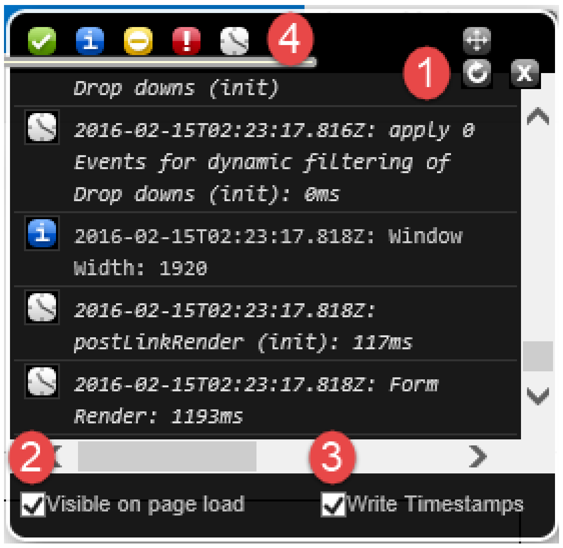
1. Update the blackbird.js files in the following directories:
   - *<Composer installation directory>\ js \ rt*
   - 15 hive of SharePoint:
     C:\Program Files\Common Files\microsoft shared\Web Server Extensions\15\TEMPLATE\LAYOUTS\js\rt
2. In each file is a function called "addMessage". Add a new line underneath the first one (content = ….), and put this in as a new line in that function:

```
content = new Date().toISOString() + ": " + content;
```

3. Save the file and close.
4. After updating each file, do a full refresh on the browser and open the Blackbird window. Time stamps should now be available.

Blackbird window overview

Winshuttle Blackbird developer tool window



```
Drop downs (init)
2016-02-15T02:23:17.816Z: apply 0 Events for dynamic filtering of Drop downs (init): 0ms
2016-02-15T02:23:17.818Z: Window Width: 1920
2016-02-15T02:23:17.818Z: postLinkRender (init): 117ms
2016-02-15T02:23:17.818Z: Form Render: 1193ms
```

✓ Visible on page load ✓ Write Timestamps

1. Refresh (circular arrow) button clears the window of any existing messages.
2. Visible on page load: Checking this box causes the Blackbird window to appear when the page is loaded.
3. Write timestamps: Checking this box adds timestamps to the beginning of each row (if available, based on the version of Composer).
4. Filter icons: Click an icon in this row to filter messages according to type (information, warning, etc.)

Tip: When using Blackbird to debug form loading issues, it is often best to do the following:

1. Load the form being tested
2. Press CTRL+F12 to open the Blackbird window
3. Clear any messages
4. Check Write Timestamps and Visible on Page Load

After this is complete, the form can be reloaded or refreshed using the appropriate icons on the browser or shortcuts on the keyboard.

These steps ensure that the messages in the window are only pertinent to the form load and do not contain messages from any previous actions. You can then review the messages in the Blackbird window to determine for any issues.

Tip: Copying the message out of the window and pasting them into a document editor such as NotePad++ often makes reviewing the messages easier to find any issues that may be present.

Common items to review in the Blackbird tool include:

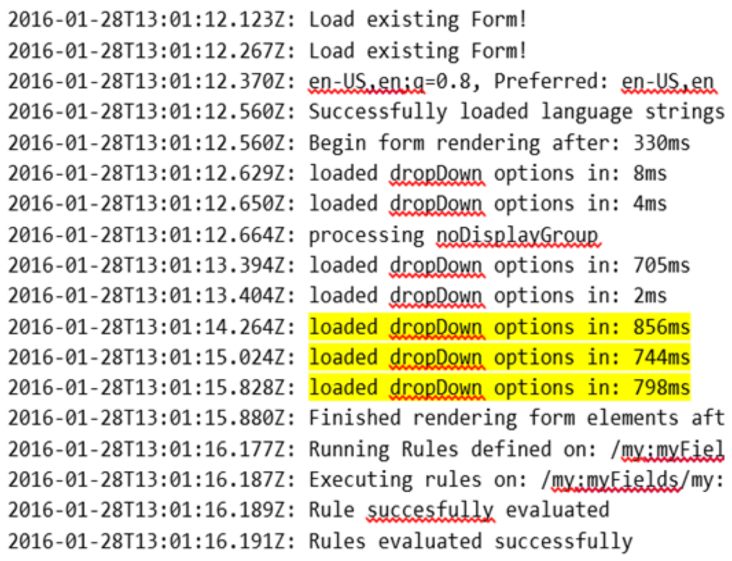
- Looking for any drop downs that are taking a long amount of time to load compared to others.
- Scanning down the timestamps to visually notice large gaps between time stamps.
- Counting the number times rules are run on a particular field. A high count could indicate circular change loops or conditions not being set correctly to minimize the number of times rules are fired.

At right: A snippet from the Blackbird window when a form was loaded.

Notice the highlighted drop down fields are taking nearly one second to load.

This is not a long time by itself but when compiled with several other drop downs that take this time, it starts to add up.

The Troubleshooting section will help address how to manage load times in cases like this.



```
2016-01-28T13:01:12.123Z: Load existing Form!
2016-01-28T13:01:12.267Z: Load existing Form!
2016-01-28T13:01:12.370Z: en-US,en;q=0.8, Preferred: en-US,en
2016-01-28T13:01:12.560Z: Successfully loaded language strings
2016-01-28T13:01:12.560Z: Begin form rendering after: 330ms
2016-01-28T13:01:12.629Z: loaded dropDown options in: 8ms
2016-01-28T13:01:12.650Z: loaded dropDown options in: 4ms
2016-01-28T13:01:12.664Z: processing noDisplayGroup
2016-01-28T13:01:13.394Z: loaded dropDown options in: 705ms
2016-01-28T13:01:13.404Z: loaded dropDown options in: 2ms
2016-01-28T13:01:14.264Z: loaded dropDown options in: 856ms
2016-01-28T13:01:15.024Z: loaded dropDown options in: 744ms
2016-01-28T13:01:15.828Z: loaded dropDown options in: 798ms
2016-01-28T13:01:15.880Z: Finished rendering form elements aft
2016-01-28T13:01:16.177Z: Running Rules defined on: /my:myFiel
2016-01-28T13:01:16.187Z: Executing rules on: /my:myFields/my:
2016-01-28T13:01:16.189Z: Rule succesfully evaluated
2016-01-28T13:01:16.191Z: Rules evaluated successfully
```

Generally, there is no single solution that will solve a problem, but these tools can assist you in determining the current state of the system and where there are opportunities for improvement, both inside and outside the Winshuttle solution.

For example, having the data that shows poor response times when pinging servers can help direct your management to make network administrator resources available to help fine tune this portion of the system while you focus on optimizing the design of your Winshuttle solution.

Troubleshooting

This section will help you troubleshoot your solutions while detailing out certain features in specific versions of Composer that will help you improve solution performance.

Every version of Winshuttle Composer has a common set of features, such as adding data connections and the ability to add custom JavaScript to enhance rules and functionality. Each version also has its own unique set of features to help tune performance. But not many customers will upgrade to the latest version of Composer.

Many of these troubleshooting options also apply to Designer-based solutions, but this section will reference Composer as the solution design tool.

The techniques discussed in this section describe various ways you can change your solution to provide quicker speeds. Keep baseline solution performance results (discussed in the previous section) in mind when implementing options in this section; you will never develop a solution that runs as fast as or faster than your baseline solution.

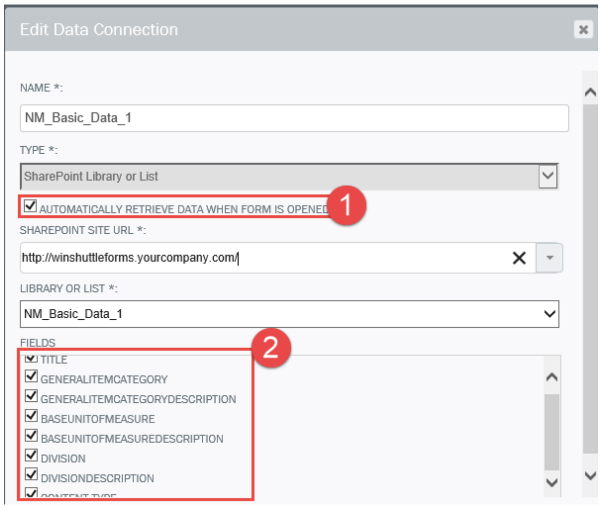
Improving Initial Load Times for forms

Limit Data Connections

Winshuttle Composer data connections (SharePoint Lists, a SQL Database, Excel files and more) can be used as sources of (or destinations for) process data. For example, you can use data connections to do things such as populate drop down fields, or to search a database.

Each data connection can impact solution performance to varying degrees.

When adding a new data connection to the solution there is the small checkbox with the text "Automatically retrieve data when form is opened".



NAME *: NM_Basic_Data_1

TYPE *: SharePoint Library or List

☑ AUTOMATICALLY RETRIEVE DATA WHEN FORM IS OPENED

SHAREPOINT SITE URL *: http://winshuttleforms.yourcompany.com/

LIBRARY OR LIST *: NM_Basic_Data_1

FIELDS
☑ TITLE
☑ GENERALITEMCATEGORY
☑ GENERALITEMCATEGORYDESCRIPTION
☑ BASEUNITOFMEASURE
☑ BASEUNITOFMEASUREDESCRIPTION
☑ DIVISION
☑ DIVISIONDESCRIPTION
☑ CONTENT TYPE

Data Connection with Retrieve on Open selected and all fields selected to be retrieved.

This setting can make a big difference to the performance of opening a form.

When this box is checked, the form will open the data connection even if the data is not yet needed when the form is opened.

For example, if a data connection is the source of a query control that runs on a field change, the data isn't needed until the query control is triggered, i.e. the user is actively interacting the form, so the data connection does not need to load when the form is opened.

Allowing data connections to be opened only when needed can drastically improve initial form load times.

Recommendations

- In most cases, this checkbox **should be checked when it is acting as the data source for a drop down-**based field. (Otherwise the data never gets loaded to the drop down and the user will have nothing to select.)
- If there are many data connections acting as the source for many drop down fields, consider changing the field type from a drop down to a text field with a Lookup control. Selecting the Lookup control calls the data connection at the time of use, so the data does not need to be retrieved when the form is loaded.
- It is a good practice to reduce the number of columns brought into the data connection to only those fields needed. This can help minimize the time required to retrieve data from the data source.

Reduce XML and HTML size

The unique combination of XML and HTML help designers collect form data, but many designers don't fully understand the relationship between XML and HTML in a solution and its impact on performance.

To view the form XML for a Winshuttle form, open the SharePoint List of Winshuttle forms. Hover the mouse over the Title column in the SharePoint list, and then click **View Item**.

When a form is opened, the entire XML of the form is opened with it. Based upon the fields designed into that view of the form, that XML is rendered into HTML to be displayed to the user.

The more XML a form contains, the longer it will take the form to load, because more XML has to be rendered into HTML.

Often when solutions are designed, fields and/or rules are used that don't work in testing, or a process is used that replaces multiple fields and/or rules. But if those unused fields are not removed from the solution, the form will take longer to load. **Eliminating any and all 'waste' from a solution helps improve performance**.

Showing and hiding fields and groups is a great feature in Composer. However, hiding unneeded fields in a view instead of removing them altogether can negatively impact form performance. Removing unused fields may not substantially reduce form loading times, but every second (or millisecond) counts.

Reduce swimlanes and improve performance for participant resolvers

Winshuttle processes attempt to resolve all swimlanes in a workflow each time the form is launched. Each resolver can also vary in how long it takes to resolve depending on what it is doing—for example, querying SharePoint or Active Directory.

> The more swimlanes in a workflow, the longer it takes to load a form.
>
> The less data the form must process to resolve a swimlane, the quicker the form processing times will be.
>
> **Removing unneeded swimlanes** from a process, or **using different methods to resolve swimlanes** can help reduce form load times.

The number of users being pulled into a swimlane can also slow performance.

For example, when using the *SiteGroupDriven* with *SelectFromRole* system control options, the swimlane will take longer to resolve as the number of people in that SharePoint group increases.

> Try to avoid pointing to lists that include **all** employees. Instead, consider using a SharePoint query with a Participant Resolver control on the form view.

Minimizing in-form processing

If a form loads quickly but takes 60 seconds for a query to return data, it still delivers a poor user experience, regardless of how much time it saves the user overall. The following sections describe how to improve in-form processing to deliver a faster, better user experience.

Use pagination for large data sets

Composer 11.0.1 introduced a pagination option for search results. Paginating data search results enables a form to show only a limited number of records at a time, reducing the time it takes to search for and display results to the form user.

This means that even though there is large data set (and thus a large XML size), the form is only rendering part of that XML, and it will only render the next set of rows when the user paginates through the result set. This can have a major impact on in-form performance with larger data sizes.

If large data sets need to be displayed in a form, consider using pagination.

Consider alternate data sources

The source of a **query** can also play a role in in-form processing time. All data sources do not perform equally.

For example, perhaps you don't want to use an SAP F4 Lookup control on a field because you want to limit the options a user can pick based on company code. The SAP F4 Lookup control provides all available options, but you cannot manipulate them. Instead you create a Winshuttle Query that runs nightly and stages the data set in a SharePoint list, where you can now apply a filter by Company Code.

What other options are there?

Query also has the ability to stage that same data in a SQL database, and it has been documented by database professionals that using SQL queries are faster than running SharePoint (CAML) queries.

Changing from SharePoint to SQL may have little to no impact on small or raw data sets, but it can impact larger data sets or data sets to which numerous filters are applied before returning the data.

Another benefit of using SQL (or other databases such as Oracle) instead of SharePoint is the ability to create **views**—a predefined data set built by a query on the server that can be used as a data source like any other table.

Using views can help reduce query time because they can hold data from many tables. They can also hold data that already has the necessary filters applied to it.

If a source SQL table contains 100,000 records and your process only needs a subset of that, your query must parse all of those records before returning the data. If you create a view in the SQL database that only contains that same subset, the query has to parse less data, and processing time decreases. Consider creating views on the SQL database to decrease in-from processing time.

Optimize Winshuttle Query

Often a query may be used to extract only one record to be used in a form – perhaps plant level details of a material in a certain plant. So how can you optimize this?

Review the Query design to determine if it is possible to search on indexed fields instead of non-indexed fields. Searching indexed fields will improve performance.

In addition, if multiple queries are being used to join multiple tables together, try using individual Query scripts for each table instead of one script that joins many tables. This can be particularly beneficial when the join between SAP table A and table B is using a mismatched join – a join where the fields do not have the same definition.

A good example of this is retrieving SAP Classification data. The classification data for many SAP objects (materials, vendors, etc.) are stored in the same tables, and the fields that are used to join to the Classification data are different.

Some SAP setups require the use of the internal object number table (INOB) to find classification data for a material. The proper joins are MARA.MATNR => INOB.CUOBJ => AUSP.OBJEK. The field INOB.CUOBJ is 18 characters while the join to AUSP.OBJEK is 40 characters. This mismatch severely affects in-form processing performance.

In this case, experience has shown that it is quicker to first query MARA and INOB to retrieve the CUOBJ value, and then use that value in a second query to retrieve the data from AUSP. Splitting the query up into multiple queries doesn't always yield faster results, but it is another tool in the toolbox.

Update Configuration Keys

Often, a form will run multiple web services consecutively with the press of a button. This can reduce page responsiveness and ultimately web service timeouts.

The web service time out default value is 500 seconds. This can be changed, but it doesn't really solve the responsiveness problem.

Winshuttle provides many configuration keys that a solution developer can use to change how a process and system run.

One such key—WebServiceProgressiveResponse—enables you to specify how a page is updated when multiple web services are run from a single button. Setting WebServiceProgressiveResponse to True forces a page to wait until all web services are completed before refreshing the page.

Setting WebServiceProgressiveResponse to False enables the page to refresh as each web service completes, which can improve response times and provide a better user experience.

Optimize rules

Form rules and their various functions vary widely. Some are purely JavaScript rules that update form field values based upon other form field values or parameters. These basic JavaScript rules generally (not always) **only run on the client side**, meaning the action uses local system resources (i.e. the user's PC). Because client-side rules do not communicate with other servers, their responsiveness is usually good.

Focusing on rules that only run on the client side will not provide much (if any) improvement unless there are errors in custom code.

Rules reliant on server interaction—for example, external queries and data lookups—can be independently optimized for performance.

Examining rules reliant on server interaction to understand when each of these process can be run is a good exercise. For example, can the rule be run at the end of the form when the user submits it, or can it run in the background by the workflow?

If a form has queries that run on form load and the form takes 10 seconds load, even if it takes the same amount of wait time, a user is more likely to report a better experience than if it takes 5 seconds to load the form and 5 seconds to submit it.

The total wait time is still 10 seconds but each wait is small. Running rules at the end may not help cumulative wait times but it could help the user experience.

Expected Results

Performance is a subjective measurement. A highly stable, integrated, and automated solution can be quickly dismissed by a user because it is not deemed fast enough.

Be creative in your design, use performance tuning tools, and troubleshoot performance issues to help you find new and different ways to achieve the same results in a faster, more streamlined solution.

Links & References

- Hardware and Software Requirements for SharePoint 2013
- IE Developer Tools
- Chrome Developer Tools – Evaluating Network Performance
- Chrome Developer Tools – Debugging Javascript

Authors

Justin Barr is a Principal Architect and CTO at Clear Process Solutions. Justin has over 16 years of IT experience specializing in systems integration and process optimization. In the past 8 years Justin has become an industry architect with SAP Winshuttle Enterprise Tools including SharePoint and Imaging automation to automate diverse processes; Material Creation, Engineer to order, Purchase Order / Capital Request, and Customer / Vendor Add/Change Request. Justin also helped define the 7 secrets to Foundation (forms and workflow) success and led the Winshuttle performance white-paper.

Lance Yoder is a Principal Architect and Partner at Clear Process Solutions. Lance is a Chemical Engineer and has over 16 years of process improvement, SAP and PMP experience. Lance is an award-winning architect of Winshuttle Studio and Foundation solutions and has helped to automate diverse processes; Material Creation, Engineer to order, Purchase Order / Capital Request, and Customer / Vendor Add/Change Request. Lance also helped define the 7 secrets to Foundation (forms and workflow) Success and co-led the development of the Winshuttle performance white-paper.

Justin and Lance work for Clear Process Solutions. More information on their firms capabilities and engagements can be found at: http://www.clearprocesssolutions.com.

WINSHUTTLE™

Microsoft Partner
Gold Portals and Collaboration
Gold Independent Software Vendor (ISV)

SAP Partner

Corporate Headquarters
Bothell, WA
Tel + 1 (800) 711-9798
Fax + 1 (425) 527-6666
www.winshuttle.com

United Kingdom
London, UK
Tel + 44 (0) 208 545 9800
Fax + 44 (0) 208 711 2665
www.winshuttle.co.uk

Germany
Bremerhaven, Germany
Tel + 49 (0) 471 142 947 0
Fax + 49 (0) 471 142 947 69
www.winshuttle-software.de

France
Maisons-Alfort, France
Tel +33 (0) 148 937 171
Fax + 33 (0) 143 683 768
www.winshuttle.fr

India
Research & Development
Chandigarh, India
Tel + 91 (0) 172 633 9800