Overview

1 How to read this documentation
    1.1 Integrating with Office Online ................................................................. 4
    1.2 What’s New .................................................................................................... 7
    1.3 Shipping your Office Online integration ...................................................... 8
    1.4 Using Trello to manage the ‘go live’ process ................................................ 11
    1.5 WOPI discovery .............................................................................................. 14
    1.6 Building a host page ....................................................................................... 20
    1.7 Office Online environments ......................................................................... 23
    1.8 Microsoft-configured settings ....................................................................... 24
    1.9 Example code ................................................................................................ 25
    1.10 WOPI Validation application ....................................................................... 26
    1.11 Testing Office Online integration ............................................................... 30
    1.12 UI guidelines ............................................................................................... 32
    1.13 Troubleshooting interactions with Office Online ......................................... 34
    1.14 Customizing Office Online using CheckFileInfo properties ........................ 38
    1.15 Using PostMessage to interact with the Office Online application iframe ... 44
    1.16 Co-authoring using Office Online ............................................................... 51
    1.17 Creating new files using Office Online ....................................................... 57
    1.18 Supporting document editing for business users ......................................... 57
    1.19 Editing binary document formats ............................................................... 59
    1.20 Verifying that requests originate from Office Online by using proof keys ... 63
    1.21 Considerations for security and privacy .................................................... 67
    1.22 Performance .................................................................................................. 68
    1.23 Frequently Asked Questions ....................................................................... 69
    1.24 Known Issues .............................................................................................. 74
    1.25 Glossary ........................................................................................................ 74
    1.26 Legal ............................................................................................................. 75
    1.27 Building this documentation locally ............................................................ 75
    1.28 Office Online Documentation Style Guide ................................................ 76
You can use the Web Application Open Platform Interface (WOPI) protocol to integrate Office Online with your application. The WOPI protocol enables Office Online to access and change files that are stored in your service.

To integrate your application with Office Online, you need to do the following:

1. Be a member of the Office 365 - Cloud Storage Partner Program. Currently integration with the Office Online cloud service is available to cloud storage partners. You can learn more about the program, as well as how to apply, at http://dev.office.com/programs/officecloudstorage.

2. Implement the WOPI protocol - a set of REST endpoints that expose information about the documents that you want to view or edit in Office Online. The set of WOPI operations that must be supported is described in the section titled requirements.

3. Read some XML from an Office Online URL that provides information about the capabilities that Office Online applications expose, and how to invoke them; this process is called WOPI discovery.

4. Provide an HTML page (or pages) that will host the Office Online iframe. This is called the host page and is the page your users visit when they open or edit Office documents in Office Online.

5. You can also optionally integrate your own UI elements with Office Online. For example, when users choose Share in Office Online, you can show your own sharing UI. These interaction points are described in the section titled Using PostMessage to interact with the Office Online application iframe.
How to read this documentation

This documentation contains an immense amount of information about how to integrate with Office Online, including details about how to implement the WOPI protocol, how Office Online uses the protocol, how you can test your integration, the process for shipping your integration, and much more. It can be difficult to know where to begin. The following guidelines can help you find the specific sections in this documentation that will be most helpful to you.

If you want to know why Office Online integration may be useful to you, and what capabilities it provides, you should read the following sections:

• Integrating with Office Online - A high level overview of the scenarios enabled by Office Online integration, as well as a brief description of some of the key technical elements in a successful integration.

• Using the WOPI protocol to integrate with Office Online - A brief description of the technical pieces that you must implement to integrate with Office Online.

If you are an engineer about to begin implementing a WOPI host, you should first read the Key concepts section. When designing your WOPI implementation, you must keep in mind the expectations around file IDs, access tokens, and locks. These concepts are critical to a successful integration with Office Online. You should also read the following sections:

• WOPI Validation application
• Troubleshooting interactions with Office Online
• Office Online environments

If you are a back-end engineer, you should begin with the following sections in addition to the Key concepts section and other general sections listed above:

• WOPI discovery
• WOPI REST endpoints
• CheckFileInfo
• Lock

Once you have read those sections, any of the other core WOPI operations are useful to read through, such as GetFile, PutFile, PutRelativeFile, UnlockAndRelock, etc.

If you are a front-end engineer, you should begin with the following sections in addition to the Key concepts section and other general sections listed above:

• Building a host page
• Using PostMessage to interact with the Office Online application iframe
• WOPI discovery, specifically the WOPI actions section
Finally, if you are looking for more details about the process for shipping your integration, see the Shipping your Office Online integration section.

1.1 Integrating with Office Online

You can integrate with Office Online to enable your users to view and edit Excel, PowerPoint, and Word files directly in the browser.

If you deliver a web-based experience that allows your users to store Office files or includes Office files as a key part of your solution, you now have the opportunity to integrate Office Online into your experience. This integration works directly against files stored by you. Your users won’t need a separate storage solution to view and edit Office files.

![Figure 1.1: Word file open for editing in Office Online](image)

1.1.1 Viewing Office files

You can make viewing available in two ways:

- By using the high-fidelity previews in Office Online as an integrated part of your experience. For example, you can use these previews in a light box view of a Word document.
- By offering to show Office files in a full-page interactive preview. Depending on your solution, this might be useful for file browsing or showing read-only files to users or in cases where users don’t have a license to edit files in Office Online.
### 1.1.2 Editing Office files

Editing is a core part of Office Online integration. When you integrate with Office Online, your users can edit Excel, PowerPoint, and Word files directly in the browser. In addition, users can edit documents collaboratively with other users using Office Online. Here are the key points to note about editing.

<table>
<thead>
<tr>
<th>Consumers</th>
<th>Business users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not need an Office 365 subscription.</td>
<td>Do need an Office 365 subscription to edit files, but not to view files.</td>
</tr>
<tr>
<td>Do not have to log on to use Office Online.</td>
<td>Are prompted to authenticate with an Office 365 or a Microsoft account to edit.</td>
</tr>
</tbody>
</table>

See also:

*Co-authoring using Office Online*  
Supporting document editing for business users

### 1.1.3 Integration process

Integrating with Office Online is relatively simple. You just need to do some HTML and JavaScript work, and set up a few simple REST endpoints. If you are familiar with existing Office protocols, note that you don’t have to implement the [MS-FSSHTTP]: File Synchronization via SOAP over HTTP Protocol (Cobalt). At a high level, to integrate with Office Online, you:

- Read XML from Office Online that describes the capabilities of Office Online. This is called **WOPI discovery**.
- Implement **REST endpoints** that Office Online uses to learn about, fetch, and update files. To do this, you implement the server side of the WOPI protocol.
- Provide an HTML page (or pages) that wrap Office Online. This page is called the **Building a host page**.

The following figure shows the WOPI protocol workflow.

To do this, you will need to ensure that your solution meets a few basic requirements.

#### Authentication

Authentication is handled by passing Office Online an access token that you generate. Assign this token a reasonable expiration date. Also, we recommend that tokens be valid for a single user against a single file, to help mitigate the risk of token leaks.

See also:

*Access token*

#### Conflict resolution

Office Online does support multiuser authoring scenarios if all users are using Office Online. However, you are responsible for managing conflicts that may come from applications other than Office Online, either with some form of file locking, or by using another type of conflict resolution.

#### File IDs

Ensure that files are represented by a persistent ID. This ID must be URL-safe because it might be passed as part of the URL at different times. Also, the ID must not change when the file is renamed, moved, or edited. This ensures an uninterrupted editing experience for your users.
See also:

File ID

File versions

You should have a mechanism by which users can clearly identify file versions through the REST APIs. Because files are cached to improve viewing performance, file versions are extremely helpful. Without them, users can’t easily determine whether they have the latest version of the file.

Desktop application integration

We encourage you to provide a way for users to open files in the Excel, PowerPoint, and Word desktop applications from the browser. Office Online is great for many things, but sometimes your users will want to transition to the desktop applications to access features that aren’t available in Office Online at this time.

1.1.4 Security considerations

Office Online is designed to work for enterprises that have strict security requirements. To make sure your integration is as secure as possible, ensure that:

- All traffic is SSL encrypted.
- Initial requests to Office Online are made by using POST, where the access token is in the body of the POST request.
Office Online identity can be established by using a public *proof key* to decrypt part of the WOPI requests. Also, the Office Online file cache indexes stored file contents by using a SHA256 hash as the cache key. You can pass Office Online the hash value using the SHA256 property in the CheckFileInfo response. If not provided, Office Online will generate a cache key from the file ID and version. To ensure that users can’t force a cache collision and view the wrong file, no user-provided information is used to generate the cache key.

### 1.1.5 Managing Office 365 subscriptions

Business users require an Office 365 subscription to edit files in Office Online. The simplest way to implement this is to have users sign in with a Microsoft account or other valid identity. This establishes that they have the correct subscription. To limit the number of times a user needs to sign in, Office Online first checks for a cookie.

To provide a better experience for users with Office 365 subscriptions, hosts can optionally implement the PutUserInfo WOPI operation. See *Tracking users’ subscription status* for more information.

### 1.1.6 Interested?

If you’re interested in integrating your solution with Office Online, take a moment to register at *Office 365 Cloud Storage Partner Program*.

### 1.2 What’s New

#### 1.2.1 Version 2016.01.27

*January 27, 2016*

- Office Online co-authoring is now supported across all WOPI hosts. See *Co-authoring using Office Online* for more information.
- Several new endpoints and operations have been added to WOPI to support Office for iOS. Some documentation is still in progress. See *WOPI REST API Reference* for more information.

**Commits**

- [Add dedicated Office for iOS docs.](https://github.com/OfficeOnline/OfficeOnlineIntegration/releases) by Tyler Butler at 2015-12-12 01:38:04
- [Reorganize CheckFileInfo properties.](https://github.com/OfficeOnline/OfficeOnlineIntegration/releases) by Tyler Butler at 2016-01-16 00:28:45

#### 1.2.2 Version 2015.12.18

*December 18, 2015*

The documentation of the WOPI REST operations has been split from the Office Online integration documentation. This was done to prepare for the expansion of the WOPI REST API that will come in a future commit.

**Commits**

- [Split narrative docs from the REST docs.](https://github.com/OfficeOnline/OfficeOnlineIntegration/releases) by Tyler Butler at 2015-11-04 23:35:31
1.2.3 Version 2015.08.03

August 3, 2015

Office Online now supports a new WOPI operation, PutUserInfo, as well as the following supporting CheckFileInfo properties:

- SupportsUserInfo
- UserInfo

These operations and properties can be used to improve the subscription verification experience for business users.

Commits

- Updated version of business editing docs. by Tyler Butler at 2015-06-12 18:47:31

1.3 Shipping your Office Online integration

Once you believe you are ready to ship your integration, you should contact Microsoft and ask to being the ‘go live’ process. This process will prepare you to ship your Office Online integration.

The ‘go live’ process consists of three phases:

1. **Validation** - Microsoft will validate your WOPI implementation as well as your UI integration. Depending on what issues are uncovered during validation, this may take some time. You should estimate a two-week turnaround time assuming there are no major issues uncovered. However, we recommend allowing at least a month unless your integration is very simple.

   **Important:** You can avoid delays in validation by ensuring that your implementation is consistent with this documentation and that the WOPI Validation application tests are passing before beginning the ‘go live’ process.

2. **Production Smoke Test** - Microsoft will enable you to use the production Office Online environment for smoke-testing. This should be basic testing performed by the host since the production environment is slightly different than the test environment, and different issues may be uncovered. This can be as long or as short as the host deems necessary. Typically a week is enough time, though it can be much shorter.

3. **Sign off and roll out** - Once Microsoft has signed off on your integration, you can begin to roll out to your users. Depending on your traffic estimates, Microsoft may request that you roll out over a period of several days to ensure you do not overload Office Online or your WOPI servers.

To manage this process, Microsoft will create a dedicated Trello board to track issues and provide a common communication channel between your team and Microsoft. You can learn more about how to use this Trello board in the Using Trello to manage the ‘go live’ process section.

Before starting the ‘go live’ process, you should read the below sections for an overview of the types of questions and issues that you may need to address during the process.

1.3.1 Preparing for the ‘go live’ process

WOPI validation

As part of the validation process, Microsoft will test your WOPI implementation using the WOPI Validation application. All of the tests in the following categories must be passing.
• HostFrameIntegration
• BaseWopi
• EditFlows
• Locks
• AccessTokens
• PutRelativeFile or PutRelativeFileUnsupported

**Note:** If you are implementing *proof key validation*, you should also check that the tests in the *ProofKeys* group are passing.

---

**Manual testing**

In addition to checking the WOPI validation results, Microsoft will do some manual validation of scenarios that cannot currently be tested using the WOPI validator. You should follow the steps in the *testing guide* and fully test these scenarios prior to starting the ‘go live’ process.

---

**Test accounts**

In order to enable your WOPI host to use Office Online’s *production environment*, Microsoft will perform some manual validation of your WOPI implementation and Office Online integration. This requires that you provide Microsoft test accounts that they can use to test your integration.

**Important:** The provided test accounts must be able to test your Office Online integration. In addition, you must provide a way for Microsoft to access the *WOPI Validation application* using these test accounts.

Once test accounts are provided, Microsoft will provide you with a rough time line to complete testing. Usually testing can be completed within two weeks. However, this time line is subject to demand; if other partners are already being tested it may take additional time for Microsoft to begin testing your implementation. In addition, if implementation issues are uncovered during testing the process may take longer.

**Business user flow test accounts**

If you are using the *business user flow*, you will need test accounts from Microsoft in order to effectively test the flow in the *Test environment*. See the *Testing the business user flow* section for more information.

**WOPI implementation questionnaire**

There are some aspects of your WOPI implementation that are particularly critical to the success of your integration. In order to verify these parts of your implementation, Microsoft will ask you to answer some questions regarding your specific WOPI implementation. These questions are included below.

**Note:** This list of questions is subject to change. Microsoft will provide you with a specific list of questions as part of the ‘go live’ process that may differ from the list below.
1. Please confirm that your File IDs meet the criteria listed in the documentation. Office Online expects file IDs to be unique and consistent over time, as well as when accessed by different users or via different UI paths (e.g. a given file might be available in two different parts of your UI, such as in a typical folder and also in search results. If the document is meant to be the same, then the file IDs should match. Otherwise users will see unexpected behavior when they access the same file via different UI paths).

2. Please confirm you’re providing a user ID using the UserId field and that the ID is unique and consistent over time as described here.

3. Please confirm that the value in the OwnerId field represents the user who owns the document and is unique and consistent over time as described here.

4. Are you sending the SHA256 value in CheckFileInfo? If not, please confirm that your version numbers change for each file version.

5. Under what conditions do you create new versions of files? Hosts often do this either when a file is unlocked or whenever a PutFile is received - either of these options is appropriate.

6. Are you using the business user flow?

7. What WOPI host capabilities properties are you passing in CheckFileInfo?

8. WOPI access tokens are currently provided in both the Authorization header and on the WOPI URL in the access_token parameter. Which of these are you using?

9. Do you use IPv6 in your datacenters?

Production settings check

Prior to enabling your integration in the production environment, Microsoft will ask you to verify your current Microsoft-configured settings, including your entries in the WOPI domain allow list and Redirect domain allow list.

Important: Remember that changes to production settings require time to make. Any changes to Microsoft-configured settings must be communicated to Microsoft by 7PM UTC on Wednesday in order to take effect in the production environment the following week.

Service management contacts

Office Online is a worldwide cloud service, and is thus monitored at all times. As part of the ‘go live’ process, Microsoft will provide you with information regarding how to escalate service quality issues with Office Online’s on-call engineers.

In order to use the production environment, you must also provide a contact for Microsoft’s on-call engineers to reach if Office Online detects an issue that we suspect is due to a problem on the host side. For example, Office Online’s monitoring systems might detect error rates for sessions spiking, and the on-call engineer would contact the host to see if it’s a known issue on the host side. Ideally this emergency contact can be reached 24x7, either by phone or email.

Rollout schedule and traffic estimates

Typically Microsoft asks partners to roll out over a period of time - between a few days to two weeks - depending on the anticipated traffic. For smaller hosts this is not always necessary. If you’re already planning on doing this, you should communicate the schedule to Microsoft (i.e. 10% day 1, 50% day 2, etc.). If you’re not, you must coordinate with Microsoft to ensure this is appropriate given your traffic estimates.
In order to best plan the rollout, you should be prepared to provide Microsoft with updated traffic estimates. Ideally these will be broken down by view/edit, file type, and geography, but provide whatever you can.

Production access

Once you and Microsoft have agreed on a rollout plan and Microsoft has signed off on your WOPI implementation, your WOPI host will be enabled in the production environment. You should plan to do some basic testing against the production environment prior to rollout to ensure there are no unique issues using that environment. Once you have completed that testing, you can roll your integration out to users according the agreed-upon rollout schedule.

1.4 Using Trello to manage the ‘go live’ process

To manage the overall ‘go live’ process, including testing and validation, Microsoft uses a service called Trello. Once you have started the ‘go live’ process, Microsoft will create a dedicated Trello board to track issues and provide a common communication channel between your team and Microsoft. The board will be pre-populated with cards tracking various questions about your WOPI implementation as well as discussion cards to determine launch dates, etc.

If you are new to Trello, you can learn more about it at https://trello.com/guide/.

**Important:** You should use the Trello board to communicate with Microsoft throughout the ‘go live’ process. This will ensure that all Office Online team members are aware of the communications, and it provides a straightforward way to isolate conversations about specific issues.

1.4.1 Adding people to the board

You can invite other relevant people to the board as needed; only people from the Office Online team and people you explicitly invite will have access to see or edit the content of the board.

We recommend that you add relevant engineers from your team to the board as well, since many of the discussions will be engineering-focused. You might add designers or business people to the board as well; simply add whomever makes sense for your team.

**See also:**

Learn how to add more people to your board at http://help.trello.com/article/717-adding-people-to-a-board.

1.4.2 Board structure

The board structure is fairly basic. There are seven lists, and you can move cards between the lists as needed. The lists serve two main purposes. First, they keep issues organized at a high level, so it is easy to see what issues are being investigated and what has been resolved, etc. In addition, the lists provide a simple way to configure Trello’s notifications such that both you and Microsoft are aware of what requires attention.

1. **Reference:** This list contains cards that have reference information, such as current test accounts for the business user flow or known issues that may affect your testing.

2. **New: Microsoft:** This list contains new cards that Microsoft needs to be aware of. You should add cards to this list to ensure it is brought to Microsoft’s attention. Any card on this list represents something that Microsoft has not yet acknowledged or taken action on. Once Microsoft is aware of the card, it will be moved to another list like Under Discussion/Investigation for action.
3. **New: Partner**: This list contains new cards that you, the Office Online partner, need to take action on. Initially, this list will contain a number of cards tracking various questions about your WOPI implementation or launch plans. As testing is done, Microsoft will create new cards to track implementation issues or additional questions that arise during testing. Like the **New: Microsoft** list, cards should be moved from this list once they are acknowledged.

4. **Under Discussion/Investigation**: This list contains cards that are being discussed or investigated, either by you or Microsoft. Once a resolution is reached on the particular card, it should be moved to the **Fix In Progress** or **Re-verify** list.

5. **Fix In Progress**: This list contains cards that are in the process of being addressed. These cards may represent a bug fix by you or Microsoft, or a settings change that is in progress, etc. Once the issue is addressed, the card should be moved to the **Re-verify** list.

6. **Re-verify**: This list contains cards that are ready to be verified. For example, you may have answered a question about your WOPI implementation, at which point you can move the card to the **Re-verify** list. Once it has been verified, it can be moved to the **Resolved** list. If there are follow-up questions or further discussion is needed, the card might be moved back to the **Under Discussion/Investigation** list.

7. **Resolved**: This list contains cards that are resolved, either because the issue has been fixed and verified, or a question has been answered and verified.

**Card flow**

With the exception of the left-most **Reference** list, the lists represent a process flow that issues will go through as they are discussed and addressed. Cards will typically move from left to right, starting at either the **New: Microsoft** or **New: Partner** lists, then moving right through the other relevant lists. In some cases, a card might be moved back to a previous list. For example, if a card in the **Re-verify** list is found to not be resolved, it may be moved back to the **Under Discussion/Investigation** or **Fix In Progress** lists.

**Tip**: You should always create new cards in either the **New: Microsoft** or **New: Partner** lists. That ensures that people
are notified about the new cards. See \textit{Notifications} for more information.

\section*{Labels}

![Default labels configured for partner boards](image)

\begin{itemize}
\item Production Blocker
\item Implementation Question
\item Launch Planning
\item Resources
\end{itemize}

\textbf{Figure 1.4: Default labels configured for partner boards}

Labels are used to help flag particular cards for easy filtering. You can filter the board based on the label colors, so it’s easy to focus on items that need to be resolved before you can be enabled in the production environment, for example, by filtering to just the red “Production Blocker” cards.

Four labels are defined initially:

1. Production Blocker
2. Implementation Question
3. Launch Planning
4. Resources

You should feel empowered to add new labels to your board if you wish.

\subsection*{1.4.3 Notifications}

Trello supports a wide variety of notification options. You can be notified of activity on your board by subscribing to individual cards, lists, or even the whole board. You’ll receive notifications when things that you’re subscribed to are changed. You can configure how these notifications behave in your Trello settings.

\textbf{Tip:} You can subscribe to an individual card yourself, but you can also be ‘added’ to a card by someone else. When you are added to a card you are automatically subscribed to it. See \url{http://help.trello.com/article/717-adding-people-to-a-board} for more information.

\textbf{See also:}

Learn how to subscribe to items in Trello at \url{http://help.trello.com/article/799-subscribing-to-cards-lists-and-boards}.

\section*{Recommended configuration and best practices}

By default, Office Online team members will subscribe to the \textit{New: Microsoft} list. This ensures that they will be notified any time a card is added or moved to that list. We recommend that your team members similarly subscribe to the \textit{New: Partner} list for the same reason.
In addition, we recommend the following:

- When you create a new card, subscribe to it so you are notified when it is updated.
- The board is pre-populated with cards. Consider subscribing to the cards that you’d like to explicitly track.
- You might also choose to subscribe to the entire board, though this can result in ‘notification overload,’ especially early on in the validation process. However, it can be useful after the board activity has lessened to ensure you don’t miss any changes.

## 1.5 WOPI discovery

WOPI discovery is the process by which a WOPI host identifies Office Online capabilities and how to initialize Office Online applications within a site. WOPI hosts use the discovery XML to determine how to interact with Office Online. The WOPI host should cache the data in the discovery XML. Although this XML does not change often, we recommend that you issue a request for the XML periodically to ensure that you always have the most up-to-date version. 12-24 hours is a good cadence to refresh although in practice it is updated much less frequently.

Another more dynamic option is to re-run discovery when proof key validation fails, or when it succeeds using the old key. That implies that the keys have been rotated, so discovery should definitely be re-run to obtain the new public key.

Finally, another option is to run discovery whenever one of your machines restarts. All of these approaches, as well as combinations of them, have been used by hosts in the past; which approach makes the most sense depends on your infrastructure.

**Important:** Hosts should not rely on the Expires HTTP header on the WOPI discovery URL in order to know when to re-run WOPI discovery. While this may change in the future, currently the value in the Expires header is not appropriate for this purpose.

**Tip:** See [WOPI discovery URLs](#) for URLs you should use to retrieve discovery XML for the Office Online test and production environments.

### 1.5.1 WOPI discovery actions

The *action* element and its attributes in the discovery XML provides some important information about Office Online.

<table>
<thead>
<tr>
<th>Element or attribute</th>
<th>Description</th>
</tr>
</thead>
</table>
| **action** element   | Represents:  
  - Operations that you can perform on an Office document.  
  - The file formats (in the form of file extensions) that are supported for the action. |
| **requires** attribute | The WOPI REST endpoints that are required to use the actions. |
| **urlsrc** attribute  | The URI that you navigate to in order to invoke the action on a particular file. |

The following example shows an *action* element in the Office Online discovery XML:
This example defines an action called edit that is supported for docx files. The edit action requires the locks and update capabilities. To invoke the edit action on a file, you navigate to the URI included in the urls value. Note that you must parse the urls value and add some parameters. For a full description of this process, see Action URLs.

Note that some actions require specific permission from Microsoft to use in the Office Online cloud service; these actions are marked . If you wish to use these actions you must contact Microsoft to have them enabled for your WOPI host.

### 1.5.2 WOPI actions

**View**
An action that renders a non-editable view of a document.

**Edit**
An action that allows users to edit a document.

**Requires** update, locks

**Editnew**
An action that creates a new document using a blank file template appropriate to the file type, then opens that file for editing in Office Online.

**Requires** update, locks

**Convert**
An action that converts a document in a binary format, such as doc, into a modern format, like docx, so that it can be edited in Office Online. See Editing binary document formats for more information about this action.

**Requires** update, locks

**Getinfo**
An action that returns a set of URLs that can be used to execute automated test cases. This action is only used by the WOPI Validation application and is meant to be used in an automated fashion.

**Interactivepreview**
An action that provides an interactive preview of the file type.

**Mobileview**
An action that renders a non-editable view of a document that is optimized for viewing on mobile devices such as smartphones.

**Tip:** Office Online automatically redirects view to mobileview when needed, so typically hosts do not need to use this action directly.

**Embedview**
An action that renders a non-editable view of a document that is optimized for embedding in a web page.

**Imagepreview**
An action that provides a static image preview of the file type.
formssubmit
An action that supports accepting changes to the file type via a form-style interface. For example, a user might be able to use this action to change the content of a workbook even if they did not have permission to use the edit action.

formedit
An action that supports editing the file type in a mode better suited to working with files that have been used to collect form data via the formssubmit action.

rest
An action that supports interacting with the file type via additional URL parameters that are specific to the file type in question.

present
An action that presents a broadcast of a document.

presentservice
This action provides the location of a broadcast endpoint for broadcast presenters. Interaction with the endpoint is described in [MS-OBPRS].

attend
An action that attends a broadcast of a document.

attendservice
This action provides the location of a broadcast endpoint for broadcast attendees. Interaction with the endpoint is described in [MS-OBPAS].

syndicate

legacywebservice

rtc

preloadedit
An action used to preload static content for Office Online edit applications.

preloadview
An action used to preload static content for Office Online view applications.

1.5.3 Action requirements

The WOPI protocol exposes a number of different REST endpoints and operations that you can perform via those endpoints. You don’t have to implement all of these for all actions. Actions define their requirements as part of the discovery XML. The requirements themselves are groups of WOPI operations that must be supported in order for the action to work.

update
Requires PutFile, PutRelativeFile

locks
Requires Lock, RefreshLock, Unlock, UnlockAndRelock

cobalt

OneNote Online Note
This requirement is only used in OneNote Online. Word, Excel, and PowerPoint Online do not require that any of the WOPI operations encompassed by this action requirement be implemented.
OneNote Online integration is not included in the Office 365 Cloud Storage Partner Program.

Requires ExecuteCellStorageRequest, ExecuteCellStorageRelativeRequest

Containers

OneNote Online Note

This requirement is only used in OneNote Online. Word, Excel, and PowerPoint Online do not require that any of the WOPI operations encompassed by this action requirement be implemented.

OneNote Online integration is not included in the Office 365 Cloud Storage Partner Program.

Requires CheckFolderInfo, DeleteFile, EnumerateChildren (folders)

1.5.4 Action URLs

The URI values provided in the urlsrc attribute in the discovery XML are not in a valid format. Simply navigating to them will result in errors. A WOPI host must transform the URLs provided in order to make them valid action URLs that can be used to invoke actions on a file. There are two key components to transforming the urlsrc attribute:

1. Parsing and replacing Placeholder values with appropriate values, or discarding them completely
2. Appending a WOPISrc value to the URI as a query string parameter

After the URL is transformed, it is a valid URL. When the URL is opened, the action will be invoked against the file indicated by the WOPISrc parameter.

Transforming the urlsrc parameter

Some WOPI actions expose parameters that hosts can use to customize the behavior of the Office Online application. For example, most actions support optional query string parameters that tell Office Online what language to render the application UI in.

These parameters are exposed in the urlsrc attribute in the discovery XML. Each of these optional parameters are contained within angle brackets (< and >), and conform to the pattern <name=PLACEHOLDER_VALUE [&]>, where name is the name of the query string parameter and PLACEHOLDER_VALUE is a value that can be replaced by the host. By convention all placeholder values in Office Online action URIs are capitalized.

The list of all placeholder values used by Office Online and what values are valid replacements for each placeholder are listed in the Placeholder values section.

The placeholders are replaced as follows:

• If the PLACEHOLDER_VALUE is unknown to the host, the entire parameter, including the angle brackets, is removed.
• Similarly, if the PLACEHOLDER_VALUE is known but the host wishes to ignore it or use the default value for that parameter, the entire parameter, including the angle brackets, should be removed.
• If the PLACEHOLDER_VALUE is known, the angle brackets are removed, the name value is left intact, and the PLACEHOLDER_VALUE string is replaced with an appropriate value. If present, the optional & must be preserved.
The following section contains a list of all current placeholder values that Office Online exposes in its discovery XML. Note that Office Online may add new placeholders and actions at any time; hosts must ignore - and thus remove from the URL per the instructions above - any placeholder values they don’t explicitly understand.

**Placeholder values**

**BUSINESS_USER**  This value can be set to 1 to indicate that the current user is a business user. This placeholder value must be used by hosts that support the business user flow. See *Supporting document editing for business users* for more information.

**DC_LLCC**  This value represents the language that Office Online should use for the purposes of data calculation. For a list of currently supported languages, see *What languages does Office Online support?* In addition to the values provided in the Locale ID column, any language can be supplied provided it is in the format described in *RFC 1766*. Typically this value should be the same as the value provided for *UI_LLCC*, and will default to that value if not provided.

**DISABLE_ASYNC**

*Note:* This value is used in the *attend* action only.

This value can be set to *true* to prevent a *broadcast* attendee from navigating a file independently.

**DISABLE_BROADCAST**

*Note:* This value is used in *broadcast* related actions only.

This value can be set to *true* to load a view of a document that does not start or join a *broadcast* session. This view looks and behaves like a regular broadcast frame.

**DISABLE_CHAT**  This value can be set to 1 to disable in-document chat functionality.

**EMBEDDED**

*Note:* This value is used in *broadcast* related actions only.

This value can be set to *true* to indicate that the output of the action will be embedded in a web page.

**FULLSCREEN**

*Note:* This value is used in *broadcast* related actions only.

This value can be set to *true* to load the file type in full-screen mode.

**PERFSTATS**

*Attention:* Sorry, this documentation hasn’t been written yet. You can track the status of *issue #52* through our public GitHub issue tracker.

**RECORDING**

*Note:* This value is used in *broadcast* related actions only.
This value can be set to `true` to load the file type with a minimal user interface.

**THEME_ID**

*Note:* This value is used in `broadcast` related actions only.

This value can be set to either 1 or 2 to designate the a specific user interface appearance. 1 denotes a light-colored theme and 2 denotes a darker colored theme.

**UI_LLCC**  This value represents the language the Office Online application UI should use. Note that Office Online does not support all languages, and may use a substitute language if the language requested is not supported. For a list of currently supported languages, see [What languages does Office Online support?](#).

In addition to the values provided in the Locale ID column, any language can be supplied provided it is in the format described in [RFC 1766](https://tools.ietf.org/html/rfc1766). If no value is provided for this placeholder, Office Online will try to use the browser language setting (`navigator.language`). If no valid language can be determined Office Online will default to English (US).

**VALIDATOR_TEST_CATEGORY**

*Note:* This value is used to run the **WOPI Validation application** in different modes.

This value can be set to All, OfficeOnline or OfficeNativeClient to activate tests specific to Office Online and Office for iOS. If omitted, the default value is All.

- **All**: activates all **WOPI Validation application** tests.
- **OfficeOnline**: activates all tests necessary for Office Online integration.
- **OfficeNativeClient**: activates all tests necessary for Office for iOS integration.

### Appending a WOPISrc value

After parsing and replacing any placeholder values in the `urlsrc` parameter, hosts must add a `WOPISrc` query string parameter to the URL. Once this is done, the URL is a valid action URL and, when loaded by a browser, will instantiate an Office Online application.

The `WOPISrc` parameter tells Office Online the URL of the host’s WOPI Files endpoint. In other words, it is a URL of the form `http://server/<...>/wopi/files/(<file_id>)`, where `file_id` is the file id of the file. The `WOPISrc` parameter value must be encoded to a URL-safe string, then the parameter is appended to the action URL.

**See also:**

See [WOPISrc](#) for more information about the `WOPISrc` value.

### Session context parameter

In addition to the `Placeholder values` listed above, hosts can optionally append an `sc` query string parameter to the action URLs. This parameter is called the session context and, if provided, will be passed back to the host in subsequent ```CheckFileInfo``` and ```CheckFolderInfo``` calls in the `X-WOPI-SessionContext` request header. There is no defined limit for the length of this string; however, since it is passed on the query string, it is subject to the overall Office Online URL length limit of 2047 bytes.
Additional notes

Depending on the specific scenario where action URLs are invoked, there are additional relevant components to action URLs. Since action URLs are typically invoked from the host page, these are covered in the Building a host page section.

1.5.5 Favicon URLs

The discovery XML includes a URL to an appropriate favicon for all Office Online applications in the favIconUrl attribute of the app element. For example:

```xml
<app name="Excel"
    favIconUrl="https://excel.officeapps.live.com/x/_layouts/resources/FavIcon_Excel.ico"
    checkLicense="true">
    ...
</app>
```

Hosts should use this URL as the favicon for their host page, so that the appropriate application icon is shown when Office Online is used.

1.6 Building a host page

In order to instantiate the Office Online applications, a host must create an HTML page that will host an iframe element within it pointing to a particular WOPI action URL. A host page provides a number of benefits, including:

1. Since the Office Online application is hosted within a host page, the host page can communicate with the frame easily using PostMessage. This allows a richer integration between the host and Office Online.
2. URLs displayed in the user’s browser are host URLs. This means that from a user’s perspective, they are interacting with the host, not Office Online. This also ensures that URLs that are copied and pasted by users are host URLs, not Office Online URLs.

The host page is typically very simple; it must meet only the following requirements:

- It must use a form element and POST the access token and access_token_ttl values to the Office Online iframe for security purposes.
- It must include any JavaScript needed to interact with the Office Online iframe using PostMessage.
- It must manage wd* query string parameters.
- It must apply some specific CSS styles to the body element and Office Online iframe to avoid visual bugs.
- It must declare a viewport meta tag to avoid visual and functional problems in mobile browsers.

1.6.1 Host page example

The Office Online GitHub repository contains a minimal example host page. Note that it does not illustrate managing wd* query string parameters or Using PostMessage to interact with the Office Online application iframe. The sections below will refer to it in more detail.
1.6.2 Passing access tokens securely

It is important, for security purposes, that access tokens not be passed to the Office Online iframe as a query string parameter. Doing so would greatly increase the likelihood of token leakage. To avoid this problem, hosts should pass the access token and access_token_ttl values to the Office Online iframe using a form POST. This technique is illustrated, along with dynamic iframe creation, in code sample 1.1.

1.6.3 Working around browser iframe behavior

Some browsers exhibit strange behavior with iframes when using bookmarks or the browser forward/back buttons. In some cases, this will cause the Office Online iframe to be loaded twice in a single navigation. This in turn can cause ‘file locked’ or ‘access token expired’ errors for users. In addition, sometimes the iframe is not recreated at all, which causes the Office Online application to load with the previous session’s state. This may cause a session to ultimately fail for a variety of reasons, including an expired CSRF (Cross-Site Request Forgery) token.

In order to work around this behavior, hosts should dynamically create the Office Online iframe using JavaScript, then dynamically submit it. This technique is illustrated in the sample host page:

```
<body>
<form id="office_form" name="office_form" target="office_frame"
    action="<OFFICE_ONLINE_ACTION_URL>" method="post">
    <input name="access_token" value="<ACCESS_TOKEN_VALUE>" type="hidden"/>
    <input name="access_token_ttl" value="<ACCESS_TOKEN_TTL_VALUE>" type="hidden"/>
</form>

<span id="frameholder"></span>

<script type="text/javascript">
    var frameholder = document.getElementById('frameholder');
    var office_frame = document.createElement('iframe');
    office_frame.name = 'office_frame';
    office_frame.id = 'office_frame';
    office_frame.title = 'Office Online Frame';

    // This attribute allows true fullscreen mode in slideshow view
    // when using PowerPoint Online's 'view' action.
    office_frame.setAttribute('allowfullscreen', 'true');

    frameholder.appendChild(office_frame);
    document.getElementById('office_form').submit();
</script>
```

Note that in an actual implementation, the <OFFICE_ONLINE_ACTION_URL>, <ACCESS_TOKEN_VALUE>, and <ACCESS_TOKEN_TTL_VALUE> strings should be replaced with appropriate action URL, access token, and access_token_ttl values.
1.6.4 Passing through wd* parameters

Office Online will sometimes pass additional query string parameters to your host page. These query string parameters are of the form wd*. When you receive these query string parameters on your host page URLs, you must pass them, unchanged, to the Office Online iframe.

In addition, if the replaceState method from the HTML5 History API is available in the user’s browser, you should remove the following parameters from your host page URL after passing them to the Office Online iframe:

- wdPreviousSession
- wdPreviousCorrelation

Other wd* parameters must not be removed from the host page URL.

1.6.5 Applying appropriate CSS styles

To ensure that the Office Online applications behave appropriately when displayed through the host page, the host page must apply some specific CSS styles to the Office Online iframe (lines 22-33) and the body element (lines 15-20) as well as set a viewport meta tag for mobile browsers (lines 11-12). In addition, the host page should set an appropriate favicon for the page using the favicon URL provided in WOPI discovery.

All of these requirements are illustrated in the sample host page:

Code sample 1.2: Markup from SampleHostPage.html illustrating appropriate styles and favicons in a host page

```html
<head>
  
  <!-- Enable IE Standards mode -->
  <meta http-equiv="x-ua-compatible" content="ie=edge">
  
  <title></title>
  
  <meta name="description" content="">
  <meta name="viewport"
    content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no">

  <link rel="shortcut icon"
    href="<OFFICE ONLINE APPLICATION FAVICON URL>" />

  <style type="text/css">
    body {
      margin: 0;
      padding: 0;
      overflow:hidden;
      -ms-content-zooming: none;
    }

    #office_frame {
      width: 100%;
      height: 100%;
      position: absolute;
      top: 0;
      left: 0;
      right: 0;
      bottom: 0;
      margin: 0;
      border: none;
  ```
1.7 Office Online environments

Office Online provides two environments for use by cloud storage partners. The WOPI discovery URLs for each environment are provided below.

Initially you’ll be given access to the test environment (also called ‘Dogfood’), and you should use that environment when building and testing your integration.

Once you believe you’re ready to release your integration, you can start the ‘Go Live’ process. As part of that process, your application will be given access to the production environment.

1.7.1 Test environment

The test environment is updated frequently - usually at least once a day - and all initial testing should be done against it. Initially this is the only environment you will be able to access.

The test environment runs the most recent Office Online code available. This does mean that you may see features there that are not yet available in production. However, the WOPI interactions between Office Online and your WOPI host should not differ dramatically between test and production. In addition, as part of the ‘Go Live’ process you’ll be given production access to verify your integration behaves as expected in that environment prior to release.

Because builds are deployed frequently to this environment, you may see regressions in behavior. However, the deployment cadence also allows us to push out changes quickly, so contact Microsoft if you experience any strange or unexpected behavior when using the test environment.

**Important:** End users should not use the test environment. If you are making Office Online integration available to end-users, you must use the production environment.

1.7.2 Production environment

The production environment is updated weekly. Once you’ve started the ‘Go Live’ process, you’ll be given production access to verify your integration behaves as expected in that environment prior to release.

1.7.3 WOPI discovery URLs

<table>
<thead>
<tr>
<th>Environment</th>
<th>Discovery URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Production</td>
<td><a href="https://onenote.officeapps.live.com/hosting/discovery">https://onenote.officeapps.live.com/hosting/discovery</a></td>
</tr>
<tr>
<td>Test/Dogfood</td>
<td><a href="https://onenote.officeapps-df.live.com/hosting/discovery">https://onenote.officeapps-df.live.com/hosting/discovery</a></td>
</tr>
</tbody>
</table>

**Tip:** These URLs are publicly accessible. However, you will not be able to invoke any WOPI actions successfully unless your WOPI domain has been added to the WOPI domain allow list.
1.8 Microsoft-configured settings

Most Office Online behavior is configurable based on properties provided in CheckFileInfo. However, there are some Office Online settings that must be changed by Microsoft.

Any changes to Microsoft-configured settings must be communicated to Microsoft by 7PM UTC on Wednesday in order to take effect in the production environment the following week.

The test environment, on the other hand, is generally updated with new settings within 24-48 hours.

1.8.1 WOPI domain allow list

Office Online only makes WOPI requests to trusted partner domains. This domain list is called the WOPI domain allow list. It contains entries of the form:

- wopi.contoso.com
- qa-wopi.contoso.com

Entries in the WOPI domain allow list are implicitly wild-carded. In other words, the entry wopi.contoso.com is equivalent to *.wopi.contoso.com. This entry will allow WOPI requests to be made to any subdomain under wopi.contoso.com.

**Tip:** If you ever generate WopiSrc values that point to a subdomain, then it needs to be on the allow list. The WopiSrc represents the domain that a WOPI client will use to execute WOPI operations against.

If you don’t ever generate WopiSrc values that point to a subdomain, then that subdomain does not need to be on the WOPI domain allow list (but it may need to be on the Redirect domain allow list).

As a general rule, hosts should use a dedicated subdomain for handling WOPI traffic. While not necessary for testing and development, this is required in production. This subdomain is typically something like wopi.hostdomain.com, though that is just a name convention and hosts can use other names if needed. This approach ensures that Office Online cannot make WOPI requests to arbitrary domains.

When you are first given access to the test environment, Microsoft will add the domains you provide to the test-only WOPI domain allow list. For test purposes you can use broadly-scoped domains such as your top-level domain (e.g. contoso.com); as part of the ‘Go Live’ process you can provide different domains for production.

**Important:** A production WOPI subdomain shouldn’t ever surface user-provided content. In other words, a user shouldn’t be able to upload something to the host and trick Office Online into making WOPI requests to the user-controlled content. That would represent a potential security hole.

1.8.2 ‘Saved to...’ name

Office Online displays a message in the bottom status bar when saving files.

![Saved to OneDrive](image)

Figure 1.5: The ‘Saved to...’ UI for OneDrive

By default, the name listed in this UI will match Office Online’s partner ID for your host. In most cases, this is the appropriate value. However, there may be cases where you wish to use a different name here than your partner ID.
For example, you may have a specific product brand that you want to display here such as ‘ContosoDrive’ instead of ‘Contoso.’ In that case, you can provide your preferred name to Microsoft.

**Important:** This value is not localized.

### 1.8.3 Redirect domain allow list

**Note:** This setting is only relevant for hosts using the *business user flow*.

When validating that a business user has an Office 365 subscription, Office Online navigates the user off of the host site so they can sign in to their Office 365 account. Once the user has signed in, Office Online will navigate back to the HostEditUrl provided by the host initially.

In order for that redirection to happen, the domain of the HostEditUrl must be on the redirect domain allow list. Like the WOPI domain allow list, entries on this list are implicitly wild-carded. The entries on this list may be different than those on the WOPI domain allow list.

Office Online also uses the DownloadUrl as part of the *business user flow*, and the domain for this URL must also be included on the redirect domain allow list.

### 1.8.4 Homepage URL

**Note:** This setting is only relevant for hosts using the *business user flow*.

As part of the *business user flow*, Office Online may determine that a user does not have the Office 365 subscription necessary for editing documents using Office Online. In this case Office Online offers to redirect the user back to the host. However, navigating the user to the HostEditUrl does not make sense in this case since the user does not have the appropriate subscription to edit, so Office Online will not load properly.

Thus, in this case Office Online navigates the user to a URL determined by the host, called the Homepage URL. This is a single setting per WOPI host and must be changed by Microsoft.

### 1.8.5 Action settings

Some WOPI *actions* and CheckFileInfo properties require special arrangements with Microsoft in order to be used. These operations are marked . If you wish to use these actions, you must contact Microsoft and request that the appropriate settings be adjusted to allow you to use these actions.

### 1.9 Example code

#### 1.9.1 Sample WOPI host

The Office Online GitHub repository contains a sample implementation of a WOPI host written in C#. This sample implementation illustrates many of the concepts necessary to implement a WOPI server, including:

- Handling requests at particular WOPI REST endpoints
• Handling WOPI operations such as CheckFileInfo, GetFile, and PutFile
• Example proof key verification (also see Proof key unit tests)

1.9.2 Proof key unit tests

Example test cases and data that can be used to validate proof key verification implementations can be found here: https://github.com/Microsoft/Office-Online-Test-Tools-and-Documentation/blob/master/samples/SampleWopiHandler/SampleWopiHandler.UnitTests/ProofKeyTests.cs

While these tests are written in C#, they can be adapted to any language. If you are having difficulties implementing proof keys, these test cases can be a useful tool for troubleshooting. See also the Troubleshooting proof key implementations section.

These same test cases along with a basic proof key validation implementation in Python is available here: https://github.com/Microsoft/Office-Online-Test-Tools-and-Documentation/blob/master/samples/python/proof_keys/tests.py

Note that the Python samples depend on PyCrypto being installed.

1.10 WOPI Validation application

The validation application provides two WOPI actions, view and getinfo, which can be used to trigger the test suite.

Warning: The test suite will test operations like PutFile, so the contents of the .wopitest file will be destroyed. In addition, some tests create new files or containers using the PutRelativeFile, CreateChildFile, and CreateChildContainer operations. While the validation application attempts to clean up these files, if there are errors in the WOPI implementation, these clean up actions may fail, leaving behind these test files. If that should happen, you must clean up these test files manually.
1.10.1 Interactive WOPI validation

The simplest way to use the validation application is to use the view action. To use the view action hosts should treat .wopitest files the same way other Office documents are treated. In other words, hosts should do the following:

1. Launch a host page pointed at the .wopitest file. Ideally, this should be the same host page used to host regular Office Online sessions. This will allow the validation application to test things like PostMessage and do some validation on the way the Office Online iframe was loaded.

2. The host page will create and navigate the Office Online iframe to the view action URL provided in WOPI discovery. The WOPIsrc and access token should be provided just like with all other actions.

3. The WOPI validation application will load and display a number of test groups. Each test group can be expanded to reveal the individual tests that it contains. You can run tests individually, by test group, or run all tests using the Run All button.

![Figure 1.6: WOPI validation application UI](image)

Tests can either pass, fail, or be skipped. Before executing any tests, Office Online will do some basic validation (e.g. confirm the file really has the .wopitest file extension) and check any applicable pre-requisites. Any test whose pre-requisites are not met will simply be skipped. For example, the tests in the EditFlows test group require the SupportsUpdate property to be set to true. If it is not, the tests in that group will all be skipped.

Once a test has been run, you can click on it to see the each request that was issued by the test and the response data. If the test failed or was skipped, the reason will be displayed just under the test name. You can click on the specific request that failed and see more information about what the test was expecting. If you are implementing proof key validation, you can use the Current Proof Key Data and Old Proof Key Data buttons to see the intermediate data on how the request was signed, which is extremely useful when debugging a proof key validation implementation.

Tip: For ease of testing, we strongly recommend that hosts support the .wopitest file extension just like all other file extensions supported by Office Online and included in WOPI discovery. This is especially important while testing.

1.10. WOPI Validation application
since it provides any user a quick and easy way to execute the validation test suite.

1.10.2 Automated WOPI validation

The WOPI Validation Application exposes a second action, getinfo. The getinfo action is designed to be used server-to-server. Instead of launching a host page, the host can simply do the following:

1. Issue a GET request to the getinfo action URL provided in WOPI discovery. The WOPIsrc, access token, and access_token_ttl should be provided just like with all other actions.

   Note: The getinfo action only supports GET requests, so the access token, and access_token_ttl values must be appended to the URL instead of being passed as POST parameters.

2. Office Online will do some basic validation (e.g. confirm the file really has the .wopitest extension) and then return a JSON-formatted array of test URLs.

3. Hosts should then make a GET request to each test URL. Office Online will run the specified test and return results in a simple JSON object. No changes to the URL are needed; the necessary parameters are included already on the URL returned from the validation application.

This is intended for automated use. For example, a host may wish to run this validation as part of rolling out new versions of their WOPI host.
Figure 1.8: Example WOPI validation results
Automated WOPI validation using a command-line tool

The host can use a Python-based command-line tool at https://github.com/Microsoft/wopi-validator-cli-python instead of launching a host page. This tool also uses the getinfo action URL provided in WOPI discovery to execute the WOPI Validation application.

1. Create a .wopitest file on the host.
2. Download and install the command-line tool by following the instructions at https://github.com/Microsoft/wopi-validator-cli-python
3. Execute the tool by providing the WOPIsrc and access token of the .wopitest file.

Tip:
1. The tool executes all the tests based on the chosen VALIDATOR_TEST_CATEGORY and displays the results.
2. To view the request and response details for each test, you can turn on verbose logging while executing the command-line tool.

1.11 Testing Office Online integration

Before starting the ‘go live’ process, you should do the following testing on your integration.

Important: You should do this testing only after the WOPI Validation application tests are passing.

1.11.1 WOPI implementation verification

The WOPI Validation application is a very useful tool in testing your WOPI implementation. However, it cannot find all potential problems in your implementation. Check the following items in addition to the results from the validator.

Non-standard CheckFileInfo properties

Hosts should avoid returning properties in CheckFileInfo that are not defined as part of the WOPI protocol. Properties may be added to CheckFileInfo in the future, and if hosts are already returning something for that property, then it might not be the expected type, or the value might not be in the expected format, etc. Thus, hosts should not return properties that aren’t in this documentation.

CheckFileInfo property return types

The WOPI Validation application tests do not currently check for the proper return types. For example, while the version value is supposed to be a string, you can return an int and the tests will pass. This can cause errors in the Office Online applications that are difficult to diagnose. For this reason, check that your return types are accurate.

Non-empty response body

Most WOPI operations do not return a response body. You should not return a response body unless the operation specifically requires it. For example, successful PutFile responses should not have a body.
1.11.2 Office Online feature verification

Many Office Online features rely on a host’s WOPI implementation. You should test the following features to help ensure your WOPI implementation is correct and that the Office Online integration is well-executed.

**Downloaded files should have most recent document changes**

If you are providing a DownloadUrl, you should ensure that a file downloaded using the Office Online Download a Copy buttons contain the most recent edits. To test this:

1. Open a document in edit mode.
2. Make an edit to the document and wait for the Saved to <HostName> text to display.
3. Click the File → Save As → Download a Copy button.
4. Check that the downloaded file has the most recent edits you made to the document.

**Download a Copy should not re-direct**

As described in DownloadUrl, Office Online expects that when directing users to the DownloadUrl, the file will be immediately downloaded. This URL should not direct the user to some separate UI to download the file.

**Rename**

If you support renaming documents within Office Online (i.e. SupportsRename is true), you should check that the rename operation behaves as expected. To test this:

1. Open a document in edit mode.
2. Click on the document name in the top title bar.
3. Rename the document.
4. Exit the Office Online application and check that the file was renamed.

If you are displaying the document name in the browser window/tab using the HTML title tag, you should check that the document name is updated after the file is renamed. If it is not, check that you are properly handling the File_Rename PostMessage.

**Save As in Excel Online**

Excel Online supports saving an open document as a new copy of that document using the File → Save As → Save As button. This feature uses the PutRelativeFile WOPI operation. You should test that this feature works as expected.

**Co-authoring**

Co-authoring support is a major boon to users, but it also provides a useful way to verify your implementation of file IDs and lock-related WOPI operations.

**Important:** The Office Online applications each have unique behavior with respect to co-authoring. Thus it is critical to test co-authoring in all three applications.
To check that co-authoring behaves as expected, you’ll need at least two different user accounts. Then, follow these steps:

1. As User A, share a document with User B.
2. Open the document in edit mode as User A.
3. Open that same document in edit mode as User B.
4. Check that both instances of the Office Online application are participating in the co-authoring session.
5. Make edits to the document as both users and ensure that both instances of the application remain connected to the co-authoring session.
6. After making some edits, leave the session and verify that the saved file contains the edits made by both User A and User B.

**Common issues**

1. If the users remain in different sessions (i.e. co-authoring does not occur) then it likely means your WOPI file IDs are not consistent. See file ID for more information.
2. If one of the users is ‘kicked out’ of the session while editing, then it likely means that you’re rejecting lock-related requests that come from a different user than the one who originally took the lock. WOPI locks are not user-owned. See Lock for more information.

**Single-user co-authoring**

While the typical co-authoring scenario is two or more users collaborating on a single document in real-time, the feature also provides other benefits as outlined in Benefits from co-authoring support.

**Important:** The Office Online applications each have unique behavior with respect to co-authoring. Thus it is critical to test co-authoring in all three applications.

To check that single-user co-authoring behaves as expected:

1. Open a document in edit mode.
2. Open a document in edit mode using the same user account originally used, but in a different browser.
3. Check that both instances of the Office Online application are participating in the co-authoring session.

**1.11.3 UI integration**

Ensure you follow the *UI guidelines* as well as the terms of the Cloud Storage Partner Program contract.

**1.12 UI guidelines**

**Important:** The guidelines here are not exhaustive. Hosts are expected to follow the terms of the Cloud Storage Partner Program contract with respect to Office Online integration.
1. **Do not display any UI on or around the Office Online editor.** The Office Online editors must always be displayed ‘edge-to-edge’, with no surrounding UI. The editor cannot be ‘light-boxed’ or integrated as a component in host UI. The editor is a standalone application. Note that the Office Online viewer can be ‘light-boxed’ or otherwise embedded in your application. However, if the user transitions to the editor, the editor must be edge-to-edge.

2. **Use Microsoft-provided application and file type icons.** See *Application and file type icons* for more information.

3. **Provide favicons for the Office Online applications.** Whenever the editor is displayed or the viewer is displayed full-window/full-tab, the favicon for the page should be set to the appropriate favicon. The preferred method is to use the URLs provided in WOPI discovery. See *Favicon URLs* for more information.

4. **Use Office Online application names in UI that activates Office Online.** For example, if you have UI in your application that reads, *Open*, this UI should read *Open in PowerPoint Online* or *Open in Office Online*.

5. **Provide support for sharing within Office Online.** Office Online provides a mechanism to share documents with other users directly within the Office Online applications. You should take advantage of this capability so that users can access sharing controls directly within Office Online. See *FileSharingPostMessage* and *FileSharingUrl* for more information.

6. **Provide breadcrumb and breadcrumb URL values.** See *Breadcrumbs* for more information.

7. **Provide an in-app Edit in Browser button.** If you are using the Office Online viewer and the current user has permissions to edit the document, you should always provide a *HostEditUrl* so that the *Edit in Browser* button is always displayed. This helps provide a more seamless transition for users.

### 1.12.1 Application and file type icons

As part of the Cloud Storage Partner Program, Microsoft will provide a ‘branding toolkit’ that includes proper file type and application icons in various sizes, as well as vector-based image formats for re-sizing.

**Tip:** The branding toolkit can be found in the Office 365 Cloud Storage Partner Program Yammer group, in the *Network Resources* section in the right sidebar. All O365 Cloud Storage partners should have access to this Yammer group.

You should use these icons as follows:

1. When displaying an Office file, either individually or as part of a list of files, use the file type icons. Do not use the application icons for this purpose.

2. When displaying a button or other UI element that opens an Office Online application, use the application icons. For example, if you display an *Open in Word Online* button, you should use the Word application icon.

**Important:** If you re-size or otherwise modify the provided icons, you must use the vector source files to maintain the high image quality of the icons.

### 1.12.2 Breadcrumbs

Breadcrumbs are an important navigational tool for users. They dramatically improve the user experience by providing helpful ‘anchors’ so users can both understand where the document they are working on is located, as well as more easily navigate in and out of the Office Online applications.

WOPI supports **two levels of breadcrumbs** only. Thus, the recommended use of these properties is as follows:
You should set these properties to the ‘root’ of your navigational hierarchy. A basic rule of thumb is that clicking this breadcrumb should take the user to their logical ‘home’ within your WOPI host.

In some cases, you may have several different siloed hierarchies within your application. In such cases it may make more sense to set these properties to the root of the particular hierarchy in which the current document is located.

Ultimately you should pick a location most appropriate for your users and application structure.

You should set these properties to the container in which the current document is located. A basic rule of thumb is that clicking this breadcrumb should take the user back to the same location they were in prior to opening the document.

Tip: If you support multiple paths to get to a file, you may wish to expose different breadcrumb properties depending on how the user navigated to the file. You can achieve this by using the Session context parameter to customize your CheckFileInfo response.

Example

Consider a logical hierarchy like this:

<table>
<thead>
<tr>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>-- Reviews</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>-- Deals</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

In this case, if the user opens Aggregate Data.xlsx, BreadcrumbBrandName/BreadcrumbBrandUrl should be set to Documents, while the BreadcrumbFolderName/BreadcrumbFolderUrl should be set to Data.

Similarly, if the user opens Integration Plans.docx, BreadcrumbBrandName/BreadcrumbBrandUrl should be set to Documents, while the BreadcrumbFolderName/BreadcrumbFolderUrl should be set to Deals.

1.13 Troubleshooting interactions with Office Online

When integrating with Office Online, it may be necessary to work with Microsoft engineers to diagnose problems. Following the steps below will help both you and Microsoft diagnose problems more quickly.

1.13.1 Before reporting issues

Before reporting any issues to Microsoft, ensure that you have done the following:

1. Check that the WOPI Validation application tests are passing. Most common issues are easily diagnosed using the validator, and passing tests are a pre-requisite for any investigations into issues you’re encountering.

Tip: In cases where the validator tests are not consistent with the documentation, assume that the validator is correct. Also please file an issue so that we can address the gaps in the documentation.
2. Check the Known Issues to see if what you’re encountering is already known. When possible, workarounds will be provided in the issue notes.

1.13.2 Fiddler traces

The most useful tool when troubleshooting Office Online integration issues is Fiddler. When you run Fiddler while reproducing an issue, it will record all HTTP requests and responses. You can then save the Fiddler trace and share it with Microsoft engineers. Fiddler traces are an invaluable tool when troubleshooting problems because they provide a full record of the HTTP traffic between the browser and Office Online. As a rule of thumb, hosts should always provide a Fiddler trace when reporting Office Online integration issues to Microsoft.

Enabling HTTPS decryption in Fiddler

Because Office Online traffic is encrypted, Fiddler must be configured to decrypt the HTTPS traffic in order to be useful. In order to enable HTTPS encryption in Fiddler, do the following:

1. From Fiddler, click Tools → Fiddler Options... to open the options dialog.
2. On the HTTPS tab, check the Capture HTTPS CONNECTs check box.
3. Check the Decrypt HTTPS traffic check box. When you do this Fiddler will display a dialog asking if you wish to trust the Fiddler Root certificate. Click Yes. You may also see some security warnings from the operating system asking if you want to install the certificate. Click Yes to all of these prompts.
4. In the drop-down, select ...from browsers only.
5. Click OK in the options dialog.
6. Close Fiddler and restart it.

Fiddler is now configured to decrypt HTTPS traffic.

Using Fiddler to trace a session

Using Fiddler to trace HTTP activity is straightforward:

1. Open Fiddler.
2. If needed, begin capturing traffic (File → Capture Traffic). Note that Fiddler starts in capture mode when it is opened, so this step may not be necessary.
3. Navigate to the host page URL while Fiddler is running, then reproduce the issue if needed.
4. Once the issue is reproduced, save the Fiddler session as an archive (File → Save → All sessions...). The resulting file should have the file extension .saz.

Using Fiddler in Linux or OS X

Fiddler works very well in Windows, but can also be used in Linux and OS X using Mono. See http://fiddler.wikidot.com/mono for more information on installing and configuring it.

1.13.3 Alternatives to Fiddler: HTTP Archives (HAR)

If you cannot use Fiddler to get session traces, you can also use the Chrome browser developer tools to save HTTP Archive (HAR) files containing the HTTP requests made by the browser. To do this, do the following:
1. Open the Chrome developer tools and select the **Network** tab.

2. Check the **Preserve log** check box if you wish to retain the request log across multiple page navigations. This makes the network tracing behave more like Fiddler, and makes it less likely that you'll lose your request log by accidentally refreshing the page or navigating away before you save the log. Office Online applications are single-page applications, so you don’t **need** to check this if you’re only planning to trace a single session.

3. After you are done reproducing the issue, right-click in the network view and select the **Save as HAR with Content** option.

4. Zip the resulting HAR file, since they can be quite large and generally compress well.

**Tip:** Other browsers’ developer tools have similar capabilities to Chrome to save session HTTP requests as an HTTP Archive.

### 1.13.4 Session IDs

Whenever an **action URL** is navigated to, Office Online creates a unique session ID. This session ID allows Microsoft engineers to quickly retrieve all server logs related to that session, including information about the WOPI calls that were made to the host. The session ID is passed back in the WOPI action URL HTTP response in the **X-UserSessionId** response header. It is also passed on every subsequent request made by the browser to Office Online in the **X-UserSessionId** request header, and it is included in all PostMessages sent from Office Online to the host page in the **wdUserSession** value.

The easiest way to retrieve the session ID is to use Fiddler, as described previously. However, you can also use the request tracking features in the Chrome and Internet Explorer developer tools to capture HTTP requests and determine
1.13. Troubleshooting interactions with Office Online

Figure 1.10: *Network* tab in the Chrome developer tools

Figure 1.11: *Save as HAR with Content* option in the Chrome developer tools
the value of the **X-UserSessionId** response header.

Full Fiddler traces are always preferred, but in cases where they’re not available, session IDs can still be used by Microsoft engineers to retrieve Office Online server logs.

### Getting session IDs after an error has occurred

In some cases, you may not be running Fiddler or browser developer tools when your session encounters an error. In these cases, the Office Online application will display an error either in a modal dialog or in a yellow bar at the top of the document below the ribbon.

Sometimes the error dialog will include the session ID in the dialog itself:

In such cases, you can copy the session ID from the error dialog.

**Tip:** Please do not simply send a screen shot of the error dialog. Copy the session ID as text and send the session ID itself to Microsoft engineers. If you send a screen shot, the Microsoft engineer will be forced to transcribe the session ID from the image, which is error-prone and tedious. Always provide the session ID as text.

In other cases, the session ID might not be available in the UI.

At this point, it is still often possible to get the session ID by using the following steps:

1. Before closing the browser, refreshing the page, or clicking any buttons in the dialog or notification bar, start Fiddler or open the browser developer tools.
2. Navigate away from the Office Online application or click a button in the dialog or notification bar.
3. You should see a request to either **WsaUpload.ashx** or **RemoteUls.ashx**. The response to those requests should include the **X-UserSessionId** header with the session ID.

#### 1.13.5 Correlation IDs

Every WOPI request Office Online makes to a host will have a unique ID, called the correlation ID. This ID will be included in the WOPI request using the **X-WOPI-CorrelationId** request header. Hosts should log this ID for each incoming WOPI request; doing so will allow hosts to easily correlate their own logs with Office Online’s server logs.

There are other WOPI request headers that may be useful for hosts to log. See the [Standard WOPI request and response headers](#) for more information.

**Tip:** In many cases, a single correlation ID is all that’s needed in order for a Microsoft engineer to retrieve complete server logs for an Office Online session for analysis. While hosts should provide Fiddler traces or session IDs whenever possible, a correlation ID will often suffice if necessary.

#### 1.14 Customizing Office Online using CheckFileInfo properties

You can customize the user interface and experience of Office Online by using a combination of **CheckFileInfo** properties as well as by implementing the **PostMessage API**.
Figure 1.12: The Chrome developer tools can be used to retrieve a session ID.

1.14. Customizing Office Online using CheckFileInfo properties
### Figure 1.13: As can the Internet Explorer developer tools.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request</td>
<td>POST /ww/WSaUpload.ashx HTTP/1.1</td>
</tr>
<tr>
<td>Accept</td>
<td>*/</td>
</tr>
<tr>
<td>X-xhr</td>
<td>1</td>
</tr>
<tr>
<td>X-UserSessionId</td>
<td>b6c23eb4-2437-45c6-b125-1f278222a</td>
</tr>
<tr>
<td>X-OfficeVersion</td>
<td>16.0.3930.3018</td>
</tr>
<tr>
<td>X-ViaCluster</td>
<td>DM28</td>
</tr>
<tr>
<td>X-Key</td>
<td>glV8Ra2tqhyFCSu2wpiYEm1lxMnFIC+BNgIal2awyaTc=,63557197621054803847</td>
</tr>
<tr>
<td>X-AccessKeyId</td>
<td>m=en-US</td>
</tr>
<tr>
<td>X-AccessTokenType</td>
<td>0</td>
</tr>
<tr>
<td>x-UserType</td>
<td>W3sIvT</td>
</tr>
<tr>
<td>Content-Type</td>
<td>application/x-www-form-urlencoded</td>
</tr>
<tr>
<td>Referer</td>
<td><a href="https://word-view.officeapps.live.com/ww/wordviewerframe.aspx?F=SD1638">https://word-view.officeapps.live.com/ww/wordviewerframe.aspx?F=SD1638</a>...</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>en-US</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>gzip, deflate</td>
</tr>
<tr>
<td>User-Agent</td>
<td>Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko</td>
</tr>
<tr>
<td>Host</td>
<td>word-view.officeapps.live.com</td>
</tr>
<tr>
<td>Content-Length</td>
<td>3228</td>
</tr>
<tr>
<td>DNT</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Office Online Integration Documentation, Release 2016.01.27**

---

Chapter 1. How to read this documentation
1.14.1 CheckFileInfo properties

CloseUrl If provided, when the Close UI is activated, Office Online will navigate the outer page (window.top.location) to the URI provided.

Hosts can also use the ClosePostMessage property to indicate a PostMessage should be sent when the Close UI is activated rather than navigate to a URL, or set the CloseButtonClosesWindow property to indicate that the Close UI should close the browser tab or window (window.top.close).

If the CloseUrl, ClosePostMessage, and CloseButtonClosesWindow properties are all omitted, the Close UI will be hidden in Office Online.

See also: CloseUrl in the WOPI REST documentation.

DownloadUrl If a DownloadUrl is not provided, Office Online will hide all UI to download the file.

If provided, Word and PowerPoint Online will display UI to download the file. When a user attempts to download the file, Word and PowerPoint will ensure that the latest document content is saved back to the WOPI host before navigating the user to the DownloadUrl to download the file.

Excel Online Note

Excel Online does not use the DownloadUrl when users click the Download a Copy button. Excel Online always downloads the file directly from the Office Online server. This has the following side effects:

1. Any content that Excel Online does not currently support, such as diagrams, are stripped from the downloaded file.
2. Excel Online does not guarantee that the latest document content is saved back to the WOPI host before downloading the file.
3. Download a Copy contains all the most recent document edits, even when the DownloadUrl is implemented incorrectly and does not point to the latest version of the document.

See also: DownloadUrl in the WOPI REST documentation.

FileSharingUrl If provided, when the Share UI is activated, Office Online will open a new browser window to the URI provided.

Hosts can also use the FileSharingPostMessage property to indicate a PostMessage should be sent when the Share UI is activated rather than navigate to a URL.

If neither the FileSharingUrl nor the FileSharingPostMessage properties are set, the Share UI will be hidden in Office Online.

See also: FileSharingUrl in the WOPI REST documentation.

HostEditUrl This URL is used by Office Online to navigate between view and edit mode.

See also: HostEditUrl in the WOPI REST documentation.
HostViewUrl  This URL is used by Office Online to navigate between view and edit mode.

See also:
HostViewUrl in the WOPI REST documentation.

SignoutUrl

If this property is not provided, no sign out UI will be shown in Office Online.

See also:
SignoutUrl in the WOPI REST documentation.

CloseButtonClosesWindow  If set to true, Office Online will close the browser window or tab (window.top.close) when the Close UI in Office Online is activated.

If Office Online displays an error dialog when booting, dismissing the dialog is treated as a close button activation with respect to this property.

Hosts can also use the CloseUrl property to indicate that the outer frame should be navigated (window.top.location) when the Close UI is activated rather than closing the browser tab or window, or set the ClosePostMessage property to indicate a PostMessage should be sent when the Close UI is activated.

If the CloseUrl, ClosePostMessage, and CloseButtonClosesWindow properties are all omitted, the Close UI will be hidden in Office Online.

Note:  The Close UI will never be displayed when using the embedview action.

See also:
CloseButtonClosesWindow in the WOPI REST documentation.

Breadcrumb properties  Office Online displays all of the Breadcrumb properties if they are provided.

1.14.2 PostMessage properties

The PostMessage properties control the behavior of Office Online with respect to incoming PostMessages. Note that if you are using the PostMessage extensibility features of Office Online, you must set the PostMessageOrigin property to ensure that Office Online accepts messages from your outer frame. You can read more about PostMessage integration at Using PostMessage to interact with the Office Online application iframe.

In cases where a PostMessage is triggered by the user activating some Office Online UI, such as FileSharingPostMessage or EditModePostMessage, Office Online will do nothing when the relevant UI is activated except send the appropriate PostMessage. Thus, hosts must accept and handle the relevant messages when the Office Online UI is triggered. Otherwise the Office Online UI will appear to do nothing when activated.

If the PostMessage API is not supported (e.g. the user’s browser does not support it, or the browser security settings prohibit it, etc.), Office Online UI that triggers a PostMessage will be hidden.

ClosePostMessage  A Boolean value that, when set to true, indicates the host expects to receive the UI_Close PostMessage when the Close UI in Office Online is activated.

Hosts can also use the CloseUrl property to indicate that the outer frame should be navigated (window.top.location) when the Close UI is activated rather than sending a PostMessage, or set the CloseButtonClosesWindow property to indicate that the Close UI should close the browser tab or window (window.top.close).

If the CloseUrl, ClosePostMessage, and CloseButtonClosesWindow properties are all omitted, the Close UI will be hidden in Office Online.
Note: The Close UI will never be displayed when using the `embedview` action.

**EditModePostMessage**  A Boolean value that, when set to `true`, indicates the host expects to receive the `UI_Edit PostMessage` when the `Edit` UI in Office Online is activated.

If this property is not set to `true`, Office Online will navigate the inner iframe URL to an edit action URL when the `Edit` UI is activated.

**EditNotificationPostMessage**  A Boolean value that, when set to `true`, indicates the host expects to receive the `Edit_Notification PostMessage`.

**FileSharingPostMessage**  A Boolean value that, when set to `true`, indicates the host expects to receive the `UI_Sharing PostMessage` when the `Share` UI in Office Online is activated.

Hosts can also use the `FileSharingUrl` property to indicate that a new browser window should be opened when the `Share` UI is activated rather than sending a PostMessage. Note that the `FileSharingUrl` property will be ignored completely if the `FileSharingPostMessage` property is set to `true`.

If neither the `FileSharingUrl` nor the `FileSharingPostMessage` properties are set, the `Share` UI will be hidden in Office Online.

**PostMessageOrigin**  A string value indicating the domain the host page will be sending/receiving PostMessages to/from. Office Online will only send outgoing PostMessages to this domain, and will only listen to PostMessages from this domain.

**Office Online Tip**

This value will be used as the `targetOrigin` when Office Online uses the HTML5 Web Messaging protocol. Therefore, it must include the scheme and host name. If you are serving your pages on a non-standard port, you must include the port as well. The literal string `*`, while supported in the PostMessage protocol, is not allowed by Office Online.

### 1.14.3 Customizing the Office Online viewer UI using CheckFileInfo

The following table describes all available buttons and UI in the Office Online viewer and what CheckFileInfo properties can be used to remove them.
### Button | How to disable
--- | ---
Edit in Browser | Two options:  
1. **(preferred)** Set `UserCanWrite` to `false` in the CheckFileInfo response (or omit it since the default for all boolean properties in CheckFileInfo is `false`)  
2. Omit the `HostEditUrl` and `EditModePostMessage` properties from the CheckFileInfo response
Share | Omit the `FileSharingUrl` and `FileSharingPostMessage` properties from the CheckFileInfo response
Download / Download as PDF | Omit the `DownloadUrl` property from the CheckFileInfo response
Print | Set the `DisablePrint` property to `true` in the CheckFileInfo response
Exit / Close | Omit the `CloseUrl` and `ClosePostMessage` properties from the CheckFileInfo response
Comments | For Word only, set the `UserCanWrite` property to `false` in the CheckFileInfo response (or omit it since the default for all boolean properties in CheckFileInfo is `false`)  
Can’t be hidden in PowerPoint
Find | Can’t be hidden
Translate | Can’t be hidden
Help | Can’t be hidden
Give Feedback | Can’t be hidden
Terms of Use | Can’t be hidden
Privacy and Cookies | Can’t be hidden
Accessibility Mode | Can’t be hidden
Start Slide Show | Can’t be hidden
Embed | Omit the `HostEmbeddedViewUrl` and `HostEmbeddedEditUrl` properties from the CheckFileInfo response
Refresh Selected Connection | Can’t be hidden
Refresh All Connections | Can’t be hidden
Calculate Workbook | Can’t be hidden
Save a Copy | Set the `UserCanNotWriteRelative` property to `true` in the CheckFileInfo response

### 1.15 Using PostMessage to interact with the Office Online application iframe

You can integrate your own UI into Office Online applications. This way, you can use your UI for actions on Office documents, such as sharing.

To integrate with Office Online in this way, implement the HTML5 Web Messaging protocol. The Web Messaging protocol, also known as PostMessage, allows the Office Online frame to communicate with its parent *host page*, and vice-versa. The following example shows the general syntax for PostMessage. In this example, `otherWindow` is a reference to another window that `msg` will be posted to.

```javascript
otherWindow.postMessage(msg, targetOrigin)
```

**Arguments**
• **msg** (*string*) – A string (or JSON object) that contains the message data.

• **targetOrigin** (*string*) – Specifies what the origin of `otherWindow` must be for the event to be dispatched. This value will be set to the `PostMessageOrigin` property provided in `CheckFileInfo`. The literal string `*`, while supported in the PostMessage protocol, is not allowed by Office Online.

### 1.15.1 Message format

All messages posted to and from the Office Online application frame are posted using the `postMessage()` function. Each message (the `msg` parameter in the `postMessage()` function) is a JSON-formatted object of the form:

```json
message

  MessageId (*string*) The name of the message being posted.

  SendTime (*long*) The time the message was sent, expressed as milliseconds since midnight 1 January 1970 UTC.

  Tip: You can get this value in most modern browsers using the `Date.now()` method in JavaScript.

Values (*JSON-formatted object*) The data associated with the message. This varies per message.
```

The following example shows the `msg` parameter for the `Host_PerfTiming` message.

```json
{
  "MessageId": "Host_PerfTiming",
  "SendTime": 1329014075000,
  "Values": {
    "Click": 1329014074800,
    "Iframe": 1329014074950,
    "HostFrameFetchStart": 1329014074970,
    "RedirectCount": 1
  }
}
```

### 1.15.2 Sending messages to the Office Online iframe

To send messages to the Office Online iframe, you must set the PostMessageOrigin property in your WOPI CheckFileInfo response to the URL of your host page. If you do not do this, Office Online will ignore any messages you send to its iframe.

You can send the following messages; all others are ignored:

• **Blur_Focus**

• **Grab_Focus**

• **Host_PerfTiming**

• **Host_PostmessageReady**

**Blur_Focus**

The Blur_Focus message signals the Office Online application to stop aggressively grabbing focus. Hosts should send this message whenever the host application UI is drawn over the Office Online frame, so that the Office application does not interfere with the UI behavior of the host.

This message only affects Office Online edit modes; it does not affect view modes.
Tip: When the host application displays UI over Office Online, it should put a full-screen dimming effect over the Office Online UI, so that it is clear that the Office application is not interactive.

Values

Empty.

Example Message:

```json
{
    "MessageId": "Blur_Focus",
    "SendTime": 1329014075000,
    "Values": { }
}
```

Grab_Focus

The Grab_Focus message signals the Office Online application to resume aggressively grabbing focus. Hosts should send this message whenever the host application UI that is drawn over the Office Online frame is closing. This allows the Office application to resume functioning.

This message only affects Office Online edit modes; it does not affect view modes.

Values

Empty.

Example Message:

```json
{
    "MessageId": "Grab_Focus",
    "SendTime": 1329014075000,
    "Values": { }
}
```

Host_PerfTiming

Provides performance related timestamps from the host page. Hosts should send this message when the Office Online frame is created so load performance can be more accurately tracked.

Values

- **Click (integer)** The timestamp, in ticks, when the user selected a link that launched the Office Online application. For example, if the host exposed a link in its UI that launches an Office Online application, this timestamp is the time the user originally selected that link.

- **Iframe (integer)** The timestamp, in ticks, when the host created the Office Online iframe when the user selected the link.

- **HostFrameFetchStart (integer)** The result of the PerformanceTiming.fetchStart attribute, if the browser supports the W3C NavigationTiming API. If the NavigationTiming API is not supported by the browser, this is 0.

- **RedirectCount (integer)** The result of the PerformanceNavigation.redirectCount attribute, if the browser supports the W3C NavigationTiming API. If the NavigationTiming API is not supported by the browser, this is 0.
Example Message:

```
{
   "MessageId": "Host_PerfTiming",
   "SendTime": 1329014075000,
   "Values": {
      "Click": 1329014074800,
      "Iframe": 1329014074950,
      "HostFrameFetchStart": 1329014074970,
      "RedirectCount": 1
   }
}
```

**Host_PostmessageReady**

Office Online delay-loads much of its JavaScript code, including most of its PostMessage senders and listeners. You might choose to follow this pattern in your WOPI host page. This means that your outer host page and the Office Online iframe must coordinate to ensure that each is ready to receive and respond to messages.

To enable this coordination, Office Online sends the `App_LoadingStatus` message only after all of its message senders and listeners are available. In addition, Office Online listens for the `Host_PostmessageReady` message from the outer frame. Until it receives this message, some UI, such as the Share button, is disabled.

Until your host page receives the `App_LoadingStatus` message, the Office Online frame cannot respond to any incoming messages except `Host_PostmessageReady`. Office Online does not delay-load its `Host_PostmessageReady` listener; it is available almost immediately upon iframe load.

If you are delay-loading your PostMessage code, you must ensure that your `App_LoadingStatus` listener is not delay-loaded. This will ensure that you can receive the `App_LoadingStatus` message even if your other PostMessage code has not yet loaded.

The following is the typical flow:

1. Host page begins loading.
2. Office Online frame begins loading. Some UI elements are disabled, because `Host_PostmessageReady` has not yet been sent by the host page.
3. Host page finishes loading and sends `Host_PostmessageReady`. No other messages are sent because the host page hasn’t received the `App_LoadingStatus` message from the Office Online frame.
4. Office Online frame receives `Host_PostmessageReady`.
5. Office Online frame finishes loading and sends `App_LoadingStatus` to host page.
6. Host page and Office Online communicate by using other PostMessage messages.

**Values**

Empty.

Example Message:

```
{
   "MessageId": "Host_PostmessageReady",
   "SendTime": 1329014075000,
   "Values": {}  
}
```
1.15.3 Listening to messages from the Office Online iframe

The Office Online iframe will send messages to the host page. On the receiving end, the host page will receive a MessageEvent. The origin property of the MessageEvent is the origin of the message, and the data property is the message being sent. The following code example shows how you might consume a message.

```javascript
function handlePostMessage(e) {
    // The actual message is contained in the data property of the event.
    var msg = JSON.parse(e.data);

    // The message ID is now a property of the message object.
    var msgId = msg.MessageId;

    // The message parameters themselves are in the Values parameter on the message object.
    var msgData = msg.Values;

    // Do something with the message here.
}
window.addEventListener('message', handlePostMessage, false);
```

The host page receives the following messages; all others are ignored:

- **App_LoadingStatus**
- **Edit_Notification**
- **File_Rename**
- **UI_Close**
- **UI_Edit**
- **UI_Sharing**

**Common Values**

In addition to message-specific values passed with each message, Office Online sends the following common values with every outgoing PostMessage:

- **ui-language** (*string*) The LCID of the language Office Online was loaded in. This value will not match the value provided using the `UI_LLCC` placeholder. Instead, this value will be the numeric LCID value (as a string) that corresponds to the language used. See [What languages does Office Online support?](#) for more information.

  This value may be needed in the event that Office Online renders using a language different than the one requested by the host, which may occur if Office Online is not localized in the language requested. In that case, the host may choose to draw its own UI in the same language that Office Online used.

- **wdUserSession** (*string*) The ID of the Office Online session. This value can be logged by host and used when troubleshooting issues with Office Online. See [Session IDs](#) for more information about this value.

- **App_LoadingStatus**
  - **Values**
    - **DocumentLoadedTime** (*long*) The time that the frame was loaded.
Example Message:

```
{
    "MessageId": "App_LoadingStatus",
    "SendTime": 1329014075000,
    "Values": {
        "DocumentLoadedTime": 1329014074983,
        "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
        "ui-language": "en-us"
    }
}
```

**Edit_Notification**

The Edit_Notification message is posted when the user first makes an edit to a document, and every five minutes thereafter, if the user has made edits in the last five minutes. Hosts can use this message to gauge whether users are interacting with Office Online. In coauthoring sessions, hosts cannot use the WOPI calls for this purpose.

To send this message, the `EditNotificationPostMessage` property in the CheckFileInfo response from the host must be set to `true`. Otherwise Office Online will not send this message.

**Values**

*Common values only.*

Example Message:

```
{
    "MessageId": "Edit_Notification",
    "SendTime": 1329014075000,
    "Values": {
        "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
        "ui-language": "en-us"
    }
}
```

**File_Rename**

The File_Rename message is posted when the user renames the current file in Office Online. The host can use this message to optionally update the UI, such as the title of the page.

**Note:** If the host does not return the `SupportsRename` parameter in their CheckFileInfo response, then the rename UI will not be available in Office Online.

**Values**

- **NewName (string)** The new name of the file.

Example Message:

```
{
    "MessageId": "File_Rename",
    "SendTime": 1329014075000,
    "Values": {
        "NewName": "Renamed Document",
        "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
        "ui-language": "en-us"
    }
}
```
UI_Close

The UI_Close message is posted when the Office Online application is closing, either due to an error or a user action. Typically, the URL specified in the CloseUrl property in the CheckFileInfo response is displayed. However, hosts can intercept this message instead and navigate in an appropriate way.

To send this message, the ClosePostMessage property in the CheckFileInfo response from the host must be set to true. Otherwise Office Online will not send this message.

Values

*Common values only.*

Example Message:

```json
{
   "MessageId": "UI_Close",
   "SendTime": 1329014075000,
   "Values": {
      "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
      "ui-language": "en-us"
   }
}
```

UI_Edit

The UI_Edit message is posted when the user activates the Edit UI in Office Online. This UI is only visible when using the view action.

To send this message, the EditModePostMessage property in the CheckFileInfo response from the host must be set to true. Otherwise Office Online will not send this message and will redirect the inner iframe to an edit action URL instead.

Hosts may choose to use this message in cases where they want more control over the user’s transition to edit mode. For example, a host may wish to prompt the user for some additional host-specific information before navigating.

Values

*Common values only.*

Example Message:

```json
{
   "MessageId": "UI_Edit",
   "SendTime": 1329014075000,
   "Values": {
      "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
      "ui-language": "en-us"
   }
}
```

UI_Sharing

The UI_Sharing message is posted when the user activates the Share UI in Office Online. The host should use this message to trigger any custom sharing UI.

To send this message, the FileSharingPostMessage property in the CheckFileInfo response from the host must be set to true. Otherwise Office Online will not send this message.
Values

Common values only.

Example Message:

```
{
  "MessageId": "UI_Sharing",
  "SendTime": 1329014075000,
  "Values": {
    "wdUserSession": "3692f636-2add-4b64-8180-42e9411c4984",
    "ui-language": "en-us"
  }
}
```

1.16 Co-authoring using Office Online

Office Online supports multiple users editing a document at the same time. Co-authoring in Office Online includes real-time content updates between all users editing the document, as well as presence information and real-time cursor tracking for each user.

There are no unique WOPI host requirements beyond those described in this documentation in order to support co-authoring. However, the guidelines around file IDs and locks, as described in the Key concepts section, are very important in order to enable co-authoring.

1.16.1 Benefits from co-authoring support

Editing documents requires that Office Online take a lock on the file to ensure that only Office Online is writing to the file. In cases where co-authoring is not supported, this causes two problems.

First, users are unable to make changes to the file while someone else is editing it. This problem is avoided when co-authoring is supported. With co-authoring in Office Online, users are never locked out of editing a document.

Second, while Office Online makes every attempt to unlock files after users have finished editing documents, there are circumstances where this may not happen. If Unlock is not called successfully, then the file will stay locked until the lock times out. This can mean that a user can be locked out of editing a file even if they were the one that originally locked it. When co-authoring is supported, the fact that the file is locked is not a problem; Office Online will allow the user to edit the document as if he or she is ‘co-authoring with him/herself’.

Thus, in addition to the obvious benefits for multi-user editing that come along with real-time co-authoring, co-authoring in Office Online provides benefits for single-user editing as well.

1.16.2 How co-authoring works in Office Online

When multiple users edit a single document using Office Online, the Office Online service manages the document changes and any necessary merges internally.

When a user attempts to edit a document, Office Online takes a lock using the Lock operation and the access token of the current user. When additional users then attempt to edit the same document, Office Online will verify those users have permission to edit the document by calling CheckFileInfo using each user’s access token. If the CheckFileInfo call succeeds and the user has permissions to edit, they will join the editing session already in progress.
Users who are collaborating will see the document content update in real-time as other users edit. However, Office Online will only call PutFile periodically with the updated document contents. There are three critical questions to consider with respect to co-authoring sessions:

1. **Auto-save frequency**: How frequently will the application call PutFile?
2. **PutFile access token**: Which user’s access token will be used when PutFile is called?
3. **Permissions check frequency**: How often will the application verify that a user still has appropriate permissions to edit the document?

Question 3 is important because PutFile is called using a single user’s access token, so a WOPI host will only be able to check permissions of the user whose access token is used. Hosts rely on Office Online to verify users still have edit permissions periodically.

The answers to these questions differ between the Office Online applications. The table below summarizes the behavior for each application, and the following sections describe this behavior in more detail.

### Code sample 1.3: Summary of co-authoring behavior for Office Online applications

<table>
<thead>
<tr>
<th>Application</th>
<th>Auto-save frequency</th>
<th>PutFile access token</th>
<th>Permissions check frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word</td>
<td>Every 30 seconds if document is updated.</td>
<td>The access token of the user who made the most recent change to the document.</td>
<td>At least every 5 minutes.</td>
</tr>
<tr>
<td>Excel</td>
<td>Every 2 minutes.</td>
<td>The access token of the user who joined the editing session most recently.</td>
<td>At least every 15 minutes.</td>
</tr>
<tr>
<td>PowerPoint</td>
<td>Every 60 seconds if document is updated.</td>
<td>The access token of the user who made the most recent change to the document.</td>
<td>At least every 5 minutes.</td>
</tr>
</tbody>
</table>

### Word Online co-authoring behavior

**Auto-save frequency**: Every 30 seconds if document is updated. In other words, if users are actively editing a document, PutFile will be called every 30 seconds. However, if users stop editing for a period of time, Word Online will not call PutFile until document changes are made again.

**PutFile access token**: For each auto-save interval, Word Online will use the access token of the user who made the most recent change to the document. In other words, if User A and B both make changes to the document within the same auto-save interval, but User B made the last change, Word Online will use User B’s access token when calling PutFile. The file will have both users’ changes, but the PutFile request will use User B’s access token.

If, on the other hand, User A made a change in one auto-save interval, and User B made a change in another auto-save interval, then Word Online will make two PutFile requests, each using the access token of the user who made the change.

**Permissions check frequency**: Word Online will verify that a user has permissions by calling CheckFileInfo at least every 5 minutes while the user is in an active session.

### Excel Online co-authoring behavior

**Auto-save frequency**: Every 2 minutes.

**PutFile access token**: Excel Online will always use the access token of the user who joined the editing session most recently. This user is called the principal user. If the principal user leaves the session, then the last user who joined the session becomes the principal user. In other words, if User A starts editing, then User A is the principal user. If User B then joins the session, User B becomes the principal user, and Excel Online will use User B’s access token when calling PutFile. The file will have both users’ changes, but the PutFile request will use User B’s access token.

If User C then joins the session, User C becomes the principal user.
If User C then leaves the session, then User B becomes the principal user, and User B’s access token will be used when calling PutFile.

Permissions check frequency: Excel Online will verify that a user has permissions by calling RefreshLock at least every 15 minutes while the user is in an active session.

PowerPoint Online co-authoring behavior

Auto-save frequency: Every 60 seconds if document is updated. In other words, if users are actively editing a document, PutFile will be called every 60 seconds. However, if users stop editing for a period of time, PowerPoint Online will not call PutFile until document changes are made again.

Note: During a single-user editing session, PowerPoint Online will only call PutFile every 3 minutes. During an active co-authoring session, that frequency is increased to every 60 seconds.

PutFile access token: For each auto-save interval, PowerPoint Online will use the access token of the user who made the most recent change to the document. In other words, if User A and B both make changes to the document within the same auto-save interval, but User B made the last change, PowerPoint Online will use User B’s access token when calling PutFile. The file will have both users’ changes, but the PutFile request will use User B’s access token.

If, on the other hand, User A made a change in one auto-save interval, and User B made a change in another auto-save interval, then PowerPoint Online will make two PutFile requests, each using the access token of the user who made the change.

Permissions check frequency: PowerPoint Online will verify that a user has permissions by calling CheckFileInfo at least every 5 minutes while the user is in an active session.

1.16.3 Scenarios

The following scenarios illustrate the behavior WOPI hosts can expect for each Office Online application when users are co-authoring.

All scenarios described here assume the following baseline flow.

Note: The pattern of WOPI calls described below is not meant to be absolutely accurate. Office Online may make additional WOPI calls beyond those described below. These scenarios are meant only to illustrate the key behavioral aspects of the Office Online applications; they are not an absolute transcript of WOPI traffic between Office Online and a WOPI host.

Scenario baseline

1. User A begins editing a document.
2. Office Online calls CheckFileInfo using User A’s access token to verify the user has edit permissions.
3. Office Online calls Lock using User A’s access token.
4. User B tries to edit the same document.
5. Office Online calls CheckFileInfo using User B’s access token to verify the user has edit permissions.
Key points

- Office Online will always verify each user has appropriate edit permissions to the document by calling CheckFileInfo using that user’s access token before allowing them to join the edit session.
- Lock will always be called using the access token of the first user to start editing the document.
- If users leave the editing session while others are still editing, Office Online will call other lock-related operations, such as Unlock or RefreshLock, using the access tokens of other users that are still editing.

Scenario 1

1. User A continues editing the document.
2. User B makes no changes.

Code sample 1.4: Co-authoring scenario 1

<table>
<thead>
<tr>
<th>Application</th>
<th>PutFile access token used</th>
<th>Additional notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Online</td>
<td>User A, since User B is not editing.</td>
<td></td>
</tr>
<tr>
<td>Excel Online</td>
<td>User B</td>
<td></td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>User A, since User B is not editing.</td>
<td></td>
</tr>
</tbody>
</table>

Scenario 2

1. User A continues editing the document.
2. User B also edits the document.

Code sample 1.5: Co-authoring scenario 2

<table>
<thead>
<tr>
<th>Application</th>
<th>PutFile access token used</th>
<th>Additional notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Online</td>
<td>Either access token may be used, depending on which user changed the document most recently during the auto-save interval.</td>
<td></td>
</tr>
<tr>
<td>Excel Online</td>
<td>User B</td>
<td></td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>Either access token may be used, depending on which user changed the document most recently during the auto-save interval.</td>
<td></td>
</tr>
</tbody>
</table>

Scenario 3

1. User A leaves the editing session by closing the Office Online application or navigating away.
2. User B continues editing the document.
3. User C tries to edit the same document.
4. Office Online calls CheckFileInfo using User C’s access token to verify the user has edit permissions.
### Scenario 4

1. User A continues editing the document.
2. User B is in the session but is not editing the document.
3. While the editing session is in progress, User B’s permissions to edit the document are removed.

**Code sample 1.7: Co-authoring scenario 4**

<table>
<thead>
<tr>
<th>Application</th>
<th>PutFile access token used</th>
<th>Additional notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Online</td>
<td>User A, since User B is not editing.</td>
<td>After no more than 5 minutes, Word Online will call CheckFileInfo with User B’s access token. That call will indicate that User B no longer has edit permissions, so User B will be removed from the editing session. User A can continue editing the document.</td>
</tr>
<tr>
<td>Excel Online</td>
<td>User B</td>
<td>Once User B’s edit permission is removed, all PutFile requests for the session will fail. In no more than 3 minutes after the first PutFile failure, then all users, including User A, will be removed from the editing session.</td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>User A, since User B is not editing.</td>
<td>After no more than 5 minutes, PowerPoint Online will call CheckFileInfo with User B’s access token. That call will indicate that User B no longer has edit permissions, so User B will be removed from the editing session. User A can continue editing the document.</td>
</tr>
</tbody>
</table>

### Scenario 5

1. User A continues editing the document.
2. User B also continues editing the document.
3. While the editing session is in progress, User B’s permissions to edit the document are removed.
### Code sample 1.8: Co-authoring scenario 5

<table>
<thead>
<tr>
<th>Application</th>
<th>PutFile access token used</th>
<th>Additional notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Online</td>
<td>Either access token may be used, depending on which user changed the document most recently during the auto-save interval.</td>
<td>If a PutFile request is made using User B’s access token, it will fail, and all users, including User A, will be removed from the editing session. If PutFile is never called with User B’s access token, then after no more than 5 minutes, Word Online will call CheckFileInfo with User B’s access token. That call will indicate that User B no longer has edit permissions, so User B will be removed from the editing session. User A can continue editing the document.</td>
</tr>
<tr>
<td>Excel Online</td>
<td>User B</td>
<td>Same as scenario 4.</td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>Either access token may be used, depending on which user changed the document most recently during the auto-save interval.</td>
<td>If a PutFile request is made using User B’s access token, it will fail, and all users, including User A, will be removed from the editing session. If PutFile is never called with User B’s access token, then after no more than 5 minutes, PowerPoint Online will call CheckFileInfo with User B’s access token. That call will indicate that User B no longer has edit permissions, so User B will be removed from the editing session. User A can continue editing the document.</td>
</tr>
</tbody>
</table>

### Scenario 6

1. User A continues editing the document.
2. User B also continues editing the document.
3. While the editing session is in progress, User A’s permissions to edit the document are removed.
1.17 Creating new files using Office Online

Hosts create new Office files using blank document templates from Office Online. In order to support this, hosts use the editnew WOPI action as follows:

1. Create a zero-byte file with the appropriate file extension (docx for Word documents, pptx for PowerPoint presentations, and xlsx for Excel workbooks).
2. Invoke the editnew action on the newly created file. Office Online will detect that the file is zero bytes based on the Size property in the CheckFileInfo response.
3. Once the editnew action has been invoked, Office Online will perform a PutFile operation on the file, overwriting it with template content appropriate to the file type. Note that this PutFile operation will be performed on an unlocked file, so hosts must ensure that PutFile operations on unlocked files that are zero bytes succeed. See the PutFile documentation for more information.

1.18 Supporting document editing for business users

Business users require an Office 365 subscription to edit files in Office Online. In order to support this scenario, Office Online requires that hosts specify that a user is a business user when using any actions that include the BUSINESS_USER placeholder value, such as edit, editnew, and view.

When business users open documents for editing, Office Online will validate that they have a valid Office 365 subscription. This may require the user to sign in using a valid Office 365 business account. This account must have an
attached Office 365 subscription that includes Office applications.

1.18.1 Indicating that a user is a business user

The first requirement in order to support document editing for business users is to indicate that the user is a business user. Doing this requires the following:

1. Set the BUSINESS_USER placeholder value on the Office Online action URL. This parameter must always be on the action URL for business users.

2. Include the LicenseCheckForEditIsEnabled property in the CheckFileInfo response. This property must always be set to true for business users.

3. Include the HostEditUrl in the CheckFileInfo response. This property must be included in the CheckFileInfo response for business users. The HostEditUrl is used to redirect the user back to the host edit page after the subscription check is completed.

4. (Optional) You may also optionally include the DownloadUrl in the CheckFileInfo response. If provided, the DownloadUrl is used to provide a direct download link to the file if the user’s subscription check fails.

Important: For security purposes, both of these URLs must be served from domains that are on the Redirect domain allow list. If either of the URLs is not on the redirect domain allow list, the flow aborts and the user is directed to Office.com.

Important: If any of the properties above are not set properly, or if the URL values provided are not on the Redirect domain allow list, then the license check flow will fail. If the flow fails, users will be redirected to Office.com.

1.18.2 Validating edit capabilities

When Office Online is loaded for business users, it will check that the user is signed in with an Office 365 business account. If the user is not signed in, they’ll be prompted to sign in.

Once signed in, Office Online will verify that the user has a valid Office 365 subscription. After this is verified, Office Online will redirect the user back to the HostEditUrl and the user can edit documents.

1.18.3 Tracking users’ subscription status

In the flow described above, the user must always be signed in with a valid Office 365 business account in order to edit documents. This is not an ideal experience since it might require the user to sign in many times.

To provide a better experience for users with Office 365 subscriptions, hosts can implement the PutUserInfo WOPI operation. Office Online will use this operation to pass back user information, including subscription status, to the host. The host can, in turn, pass the UserInfo string back to Office Online on subsequent CheckFileInfo responses for that user. Office Online will use the data in the UserInfo string to determine whether a subscription check is needed, and in most cases will not require the user to sign in. Note that hosts must treat the UserInfo string as an opaque string.

Important: Hosts must treat the UserInfo string as an opaque string.

This approach helps ensure that users are required to sign in to validate their Office 365 subscription as infrequently as possible.
1.18.4 Testing the business user flow

In order to test the business user flow in the *Test environment*, you must use test Office 365 user accounts provided by Microsoft. These accounts are provided in the Cloud Storage Partner Program Yammer group.

These test accounts are periodically rotated. If you have trouble signing in while testing the business user flow, check that the accounts you’re using are the most recent ones provided.

1.19 Editing binary document formats

**Pre-release Feature**

This documentation is for an upcoming feature and may undergo dramatic changes prior to final release. Pre-release features are available in the *Test environment* only.

Office Online does not support editing files in binary formats such as doc, ppt, and xls, directly. However, Office Online can convert documents in those formats to modern formats like docx, pptx, and xlsx, so that users can then edit them in Office Online.

**Important:** Conversion is almost always a lossless process, and there are typically very few, if any, user-visible changes to the layout and formatting of the document during conversion. However, this is not always the case, and hosts should be aware that users may wish to revert back to the previous binary version of the document after it has been converted.

The conversion capability is exposed in WOPI discovery as the `convert` action. In order to support editing binary file formats, hosts must support the `PutRelativeFile` WOPI operation. The conversion process is started by simply invoking the `convert` action on the binary file. The high level process is as follows:

1. The host invokes the `convert` action on a binary file.
2. Office Online will retrieve the file from the host and convert it.
3. Office Online will send the converted document back to the host by issuing a `PutRelativeFile` request against the original file ID.
4. Hosts can use the `X-WOPI-FileConversion` header on the `PutRelativeFile` request to determine that the request is being made in the context of a file conversion. Hosts can thus treat these requests differently than other `PutRelativeFile` requests.
5. After the document is successfully converted, Office Online will redirect the user to the `HostEditUrl` returned in the `PutRelativeFile` response.

1.19.1 Enabling ‘convert and edit’ from within the Office Online viewer

In the basic conversion flow described above, a user will invoke the `convert` action using some UI on the host. However, hosts may wish to open a document first in the Office Online viewer and use Office Online’s in-application Edit button to convert and edit the document, as is done with documents in editable formats.

Hosts can support this in the same way that view -> edit transitions are typically supported. Hosts must do the following when opening binary documents using the `view` action:

1. Set the `UserCanWrite` property to `true`.
2. Set the `UserCanNotWriteRelative` property to `false` (or simply omit it).
3. Set the HostEditUrl property to a host URL that will invoke the convert action when loaded.

Following these steps will ensure that the in-application Edit button is displayed. When clicked, this button will navigate the user to the HostEditUrl provided for the binary file, which will in turn begin the conversion process and eventually redirect the user to the HostEditUrl for the newly converted document.

Hosts may optionally handle the in-application Edit button themselves by setting theEditModePostMessage property to true and handling the UI_Edit PostMessage.

1.19.2 Customizing the conversion process

In the basic conversion process, Office Online will create a new file each time a user attempts to edit a file in a binary file format. For example, consider this scenario:

1. A user opens a binary file named File.doc in the Office Online viewer.
2. The user clicks the Edit button in the Office Online viewer.
3. The conversion process is started, and Office Online calls PutRelativeFile on the host, creating a newly converted file, File.docx.
4. The user edits the newly converted document, then ends the editing session.
5. Later, the user returns and opens the original binary file, File.doc, in the Office Online viewer.

At this point, the user may be confused as to why the changes made earlier are not in the document. If the user attempts to edit the file again, Office Online will again convert it and create a second converted file, for example File1.docx.

This can be very confusing for users depending on how the user experience within the host UI is designed. Thus, it is important to consider how to manage user confusion around converted documents. There are three basic customization options that hosts can employ to help manage this.

First, the host can choose to display some UI to the user prior to beginning the conversion process. Because hosts ultimately control when the convert action is invoked, a host could choose to display a notification message when a user attempts to edit a binary document, informing them that the document will be converted. This can also apply to the in-application Edit button by setting the EditModePostMessage property to true and handling the UI_Edit PostMessage.

Second, the host can choose to handle converted documents in a unique way, by handling the PutRelativeFile operation differently when called from the conversion flow. The X-WOPI-FileConversion header tells hosts when the operation is being called from the conversion flow, so the host can choose how best to handle those requests.

Finally, the host can control where the user is navigated after conversion is complete. Office Online navigates to the HostEditUrl that is returned in the PutRelativeFile response, which the host controls. Thus, hosts can customize where the user lands after the conversion is finished. This allows hosts to opt not to send the user directly to the Office Online editor, but to any URL they wish. For example, a host may redirect the user to an interstitial page that informs them their document has been converted.

The following are some examples illustrating how these options can be used by hosts to change the user experience around file conversion. Note that these examples are not meant to be exhaustive, and that hosts may opt to customize the conversion process and flow in ways not described here.

Example 1

In the following example, the host helps the user understand the conversion process by naming the converted file such that it is clear that it was converted from a binary file.

1. A user selects a binary file in the host UI and chooses to edit it using Office Online.
2. The conversion process is started, and Office Online calls PutRelativeFile with the converted document content.
3. The host creates a new file as part of the PutRelativeFile request and appends (Editable) to the name of the file.

4. The user is navigated to a page that allows them to edit the newly converted file in Office Online.

Example 2

In the following example, the host wishes to hide the conversion process from the user to provide the most frictionless experience possible.

1. A user selects a binary file in the host UI and chooses to edit it using Office Online.

2. The conversion process is started, and Office Online calls PutRelativeFile with the converted document content.

3. Rather than create a new file, the host chooses to add the converted file as a new version to the existing binary file.

4. The user is navigated to a page that allows them to edit the newly converted file in Office Online.

5. The user can restore the binary version of the file by using the ‘version history’ features within the host.

Note: This approach may not be feasible for all hosts, depending on how file metadata and versions are handled within their system. However, it does offer the following benefits:

- The user only ever sees a single document both before and after the document is converted.
- Since there is always only a single document, the user always finds the ‘right’ document. That is, if the user edited the file - which is likely since they invoked the conversion process by attempting to edit a binary document - then when they open the file a second time, their previous edits will be there, just as they expect.

Example 3

In the following example, the host has deemed it important to inform users explicitly about the conversion process and its possible side effects.

1. A user selects a binary file in the host UI and chooses to edit it using Office Online.

2. The host displays a notification message with the following text:

   In order to edit File.doc, it must be converted to a modern file format. If the document doesn’t look the same after it’s converted, don’t worry - you can always get back to the original file if you need to.

   The user can cancel the conversion operation or choose to continue with it.

3. If the user chooses to continue, the host navigates them to a page that invokes the convert action on the file.

4. The conversion process is started, and Office Online calls PutRelativeFile with the converted document content.

5. The host returns a special URL in the HostEditUrl property in the PutRelativeFile response. Office Online navigates the user to that URL once the conversion is complete.

6. The user lands on the URL specified by the host, and sees the following message:

   Your file, File.doc, has been converted to a new file, File.docx. The new file is in a modern file format, and the file extension has changed. If you don’t need the original file any more, you can delete it.

   The message includes a button that the user can use to delete the original file immediately if they wish.

7. Once the user clicks OK, they’re navigated to a page that invokes the edit action on the converted file.
Convert and edit File.doc?

In order to edit File.doc, it must be converted to a modern file format. If the document doesn't look the same after it's converted, don't worry - you can always get back to the original file if you need to.

Figure 1.14: Example conversion notification message

File has been converted for editing

Your file, File.doc, has been converted to a new file, File.docx. The new file is in a modern file format, and the file extension has changed. If you don't need the original file any more, you can delete it.

Figure 1.15: Example conversion completed message
Variant 3.1: Display post-conversion message in the Office Online UI

In steps 5 and 6, rather than navigating the user to an interstitial page, the host may choose to append some parameters to the standard HostEditUrl. Then, when that HostEditUrl is navigated to, the host page can use the parameters that were added to the URL to determine that the dialog described in step 6 should be displayed. The host can display that notification above the Office Online editor frame. This is similar to what hosts do when handling the `UI_Sharing_PostMessage`.

**Tip:** Hosts must ensure that they properly use the `Blur_Focus` and `Grab_Focus` messages when drawing UI over the Office Online frame.

### 1.20 Verifying that requests originate from Office Online by using proof keys

When processing WOPI requests from Office Online, you might want to verify that these requests are coming from Office Online. To do this, you use proof keys.

Office Online signs every WOPI request with a private key. The corresponding public key is available in the `proof-key` element in the WOPI discovery XML. The signature is sent with every request in the `X-WOPI-Proof` and `X-WOPI-ProofOld` HTTP headers.

The signature is assembled from information that is available to the WOPI host when it processes the incoming WOPI request. To verify that a request came from Office Online, you must:

- Create the expected value of the proof headers.
- Use the public key provided in WOPI discovery to decrypt the proof provided in the `X-WOPI-Proof` header.
- Compare the expected proof to the decrypted proof. If they match, the request originated from Office Online.
- Ensure that the `X-WOPI-TimeStamp` header is no more than 20 minutes old.

**Note:** Requests to the `FileUrl` will not be signed. The `FileUrl` is used exactly as provided by the host, so it does not necessarily include the access token, which is required to construct the expected proof.

**Tip:** The [Office Online GitHub repository](https://github.com/OfficeOnline/office-online-sdk-cs) contains a set of unit tests that hosts can adapt to verify proof key validation implementations. See [Proof key unit tests](https://github.com/OfficeOnline/office-online-sdk-cs/blob/master/Tests/ProofKey/ProofKey.cs) for more information.

#### 1.20.1 Constructing the expected proof

To construct the expected proof, you must assemble a byte array consisting of the access token, the URL of the request (in uppercase), and the value of the `X-WOPI-TimeStamp` HTTP header from the request. Each of these values must be converted to a byte array. In addition, you must include the length, in bytes, of each of these values.

To convert the access token and request URL values, which are strings, to byte arrays, you must ensure the original strings are in UTF-8 first, then convert the UTF-8 strings to byte arrays. Convert the `X-WOPI-TimeStamp` header to a `long` and then into a byte array. Do not treat it as a string.

Then, assemble the data as follows:

- 4 bytes that represent the length, in bytes, of the `access_token` on the request.
Office Online Integration Documentation, Release 2016.01.27

- The access_token.
- 4 bytes that represent the length, in bytes, of the full URL of the WOPI request, including any query string parameters.
- The WOPI request URL in all uppercase. All query string parameters on the request URL should be included.
- 4 bytes that represent the length, in bytes, of the X-WOPI-TimeStamp value.
- The X-WOPI-TimeStamp value.

The following C# code snippet illustrates the construction of an expected proof.

Code sample 1.10: Sample code from ProofKeyHelper.cs

```csharp
public bool Validate(ProofKeyValidationInput testCase)
{
    // Encode values from headers into byte[]
    var accessTokenBytes = Encoding.UTF8.GetBytes(testCase.AccessToken);
    var hostUrlBytes = Encoding.UTF8.GetBytes(testCase.Url.ToUpperInvariant());
    var timeStampBytes = EncodeNumber(testCase.Timestamp);

    // prepare a list that will be used to combine all those arrays together
    List<byte> expectedProof = new List<byte>(
        4 + accessTokenBytes.Length +
        4 + hostUrlBytes.Length +
        4 + timeStampBytes.Length);

    expectedProof.AddRange(EncodeNumber(accessTokenBytes.Length));
    expectedProof.AddRange(accessTokenBytes);
    expectedProof.AddRange(EncodeNumber(hostUrlBytes.Length));
    expectedProof.AddRange(hostUrlBytes);
    expectedProof.AddRange(EncodeNumber(timeStampBytes.Length));
    expectedProof.AddRange(timeStampBytes);

    // create another byte[] from that list
    byte[] expectedProofArray = expectedProof.ToArray();

    // validate it against current and old keys in proper combinations
    bool validationResult =
        TryVerification(expectedProofArray, testCase.Proof, _currentKey.CspBlob) ||
        TryVerification(expectedProofArray, testCase.OldProof, _currentKey.CspBlob) ||
        TryVerification(expectedProofArray, testCase.Proof, _oldKey.CspBlob);

    // TODO:
    // in real code you should also check that TimeStamp header is no more than 20 minutes old
    // but because we're using predefined test cases to validate that the method works
    // we can't do it here.
    return validationResult;
}
```

1.20.2 Retrieving the public key

Office Online provides two different public keys as part of the WOPI discovery XML: the current key and the old key. Two keys are necessary because the discovery data is meant to be cached by the host, and Office Online periodically rotates the keys it uses to sign requests. When the keys are rotated, the current key becomes the old key, and a new current key is generated. This helps to minimize the risk that a host does not have updated key information from WOPI discovery when Office Online rotates keys.
Both keys are represented in the discovery XML in two different formats. One format is for WOPI hosts that use the .NET framework. The other format can be imported in a variety of different programming languages and platforms.

**Using .NET to retrieve the public key**

If your application is built on the .NET framework, you should use the contents of the value and oldvalue attributes of the proof-key element in the WOPI discovery XML. These two attributes contain the Base64-encoded public keys that are exported by using the RSACryptoServiceProvider.ExportCspBlob method of the .NET Framework.

To import this key in your application, you must decode it from Base64 then import it by using the RSACryptoServiceProvider.ImportCspBlob method.

**Using the RSA modulus and exponent to retrieve the public key**

For hosts that don’t use the .NET framework, Office Online provides the RSA modulus and exponent directly. The modulus and exponent of the current key are found in the modulus and exponent attributes of the proof-key element in the WOPI discovery XML. The modulus and exponent of the old key are found in the oldmodulus and oldexponent attributes. All four of these values are Base64-encoded.

The steps to import these values differ based on the language, platform, and cryptography API that you are using.

The following example shows how to import the public key by using the modulus and exponent in a Python program using the PyCrypto library.

**Code sample 1.11: Generating a public key from a modulus and exponent in Python**

```python
def generate_key(modulus_b64, exp_b64):
    """
    Generates an RSA public key given a base64-encoded modulus and exponent
    :param modulus_b64: base64-encoded modulus
    :param exp_b64: base64-encoded exponent
    :return: an RSA public key
    """
    mod = int(b64decode(modulus_b64).encode('hex'), 16)
    exp = int(b64decode(exp_b64).encode('hex'), 16)
    seq = asn1.DerSequence()
    seq.append(mod)
    seq.append(exp)
    der = seq.encode()
    return RSA.importKey(der)
```

**1.20.3 Verifying the proof keys**

After you import the key, you can use a verification method provided by your cryptography library to verify incoming requests were signed by Office Online. Because Office Online rotates the current and old proof keys periodically, you have to check three combinations of proof key values:

- The X-WOPI-Proof value using the current public key
- The X-WOPI-ProofOld value using the current public key
- The X-WOPI-Proof value using the old public key

If any one of the values is valid, the request was signed by Office Online.

The following example shows how to verify one of these combinations in .NET using C#.
private static bool TryVerification(byte[] expectedProof, string signedProof, string publicKeyCspBlob)
{
    using (RSACryptoServiceProvider rsaAlg = new RSACryptoServiceProvider())
    {
        byte[] publicKey = Convert.FromBase64String(publicKeyCspBlob);
        byte[] signedProofBytes = Convert.FromBase64String(signedProof);
        try
        {
            rsaAlg.ImportCspBlob(publicKey);
            return rsaAlg.VerifyData(expectedProof, "SHA256", signedProofBytes);
        }
        catch (FormatException)
        {
            return false;
        }
        catch (CryptographicException)
        {
            return false;
        }
    }
    return false;
}

The following example shows how to verify one of these combinations in Python using the PyCrypto library.

def try_verification(expected_proof, signed_proof_b64, public_key):
    """
    Verifies the signature of a signed WOPI request using a public key provided in
    WOPI discovery.
    :param expected_proof: a bytearray of the expected proof data
    :param signed_proof_b64: the signed proof key provided in the X-WOPI-Proof or
    X-WOPI-ProofOld headers. Note that the header values are base64-encoded, but
    will be decoded in this method
    :param public_key: the public key provided in WOPI discovery
    :return: True if the request was signed with the private key corresponding to
    the public key; otherwise, False
    """
    signed_proof = b64decode(signed_proof_b64)
    verifier = PKCS1_v1_5.new(public_key)
    h = SHA256.new(expected_proof)
    v = verifier.verify(h, signed_proof)
    return v

1.20.4 Troubleshooting proof key implementations

If you are having difficulty with your proof key verification implementation, here are some common issues that you
should investigate:

- Verify you’re converting the URL to uppercase.
- Verify you’re including any query string parameters on the URL when transforming it for the purposes of
  building the expected proof value.
• Verify you’re using the same encoding for any special characters that may be in the URL.

• Verify you’re using an HTTPS URL if your WOPI endpoints are HTTPS. This is especially important if you have SSL termination in your network prior to your WOPI request handlers. In this case, the URL Office Online will use to sign the requests will be HTTPS, but the URL your WOPI handlers ultimately receive will be HTTP. If you simply use the incoming request URL your expected proof will not match the signature provided by Office Online.

In addition, use the Proof key unit tests to verify your implementation with sample data.

1.21 Considerations for security and privacy

1.21.1 Office Online and customer data

There are two classes of customer data that flow through Office Online servers: user metadata and customer content (documents, presentations, workbooks, and notebooks).

User metadata

User metadata consists of URLs, email addresses, user IDs, etc. This data lives in memory and travels back and forth between Office Online and the WOPI host through HTTPS. Office Online goes to great lengths to scrub all personally identifying information (PII) from its logs. This is regularly audited to ensure Office Online is compliant with several different privacy standards (such as FedRamp in the USA).

Customer content

In the case of customer content, Office Online retrieves it from the host in order to render it for viewing or editing. In the image below you can see how WOPI is used to fetch content from the host and send content changes to the host (ContosoDrive in this case).

With the exception of caching (discussed below), it is reasonable to say that customer content only lives on Office Online servers during a user session. That is, a user can reasonably expect that when they end an editing session, their content will no longer live anywhere on Office Online servers once it has been saved to the host. The key exception here is the Office Online viewing cache.

The Office Online viewing cache

In order to optimize view performance for PowerPoint Online and Word Online, Office Online stores rendered documents in a local disk cache. This way, if more than one person wants to view a document, Office Online only has to fetch it and render it once. It is important to note that a complete removal of the cache would significantly degrade the customer experience for many users and dramatically increase the cost of running Office Online.

Documents in the cache are indexed using a SHA256 hash that is generated based on the contents of the file (or some other unique attribute of the file). No user information is used to index the file. On every request for a file, if the WOPI host validates the access token, Office Online uses the SHA256 hash returned by the host (or generated based on file id + version value) to check for the file in the document cache. Since the SHA256 hash is an enormous number that is generated using information that is unique to the file, there is essentially no chance of the same number being generated twice. Also, since the hash is generated using information unique to the file and not based on any sort of user data, Office Online cannot retrieve information that is specific to a given user. Office Online specifically does not log the SHA256 hash when the file is cached so that it is effectively impossible for Office Online to retrieve information associated with a specific host or user without the participation of the host.
Documents live in the cache until they become unpopular. That is, the cache is not time-based but rather is based on available space and usage. Unpopular files may expire out of the cache in only a few days while popular documents may remain in the cache for up to 30 days.

As of May 2016 the contents of the cache is encrypted using a FIPS 140-2 compliant encryption algorithm.

### 1.22 Performance

#### 1.22.1 Preloading static content

One way to improve the load time performance of Office Online applications is to preload Office Online’s static content (JavaScript, CSS, and images) into the user’s browser cache. This will help ensure that when the user opens a document in Office Online, they can use the previously cached static content and do not need to download that data when they first try to load Office Online.

To support preloading static content, Office Online provides two WOPI actions for each Office Online application in its discovery XML, one to preload static content for the view action (preloadview), and a second to preload static content for the edit action (preloadedit).

Hosts can use these URLs just like they use other Action URLs, by pointing iframes in their pages at the action URL. Unlike most action URLs, the WopiSrc and access token do not need to be specified in order to use these actions.

Hosts can include both preloadview and preloadedit in their pages to preload static content for both. Note that the static content preload actions contain the UI_LLCC placeholder value, which should be replaced with an appropriate language for the user so that the proper localized static content is preloaded.
Tip: If you wish to preload static content for all applications and both view and edit modes, you must load multiple actions, one for each application/mode combination.

1.22.2 Optimizing document viewing for high volume

Attention: Sorry, this documentation hasn’t been written yet. You can track the status of issue #5 through our public GitHub issue tracker.

1.23 Frequently Asked Questions

1.23.1 Why does Office Online pass the access token in both the Authorization HTTP header and as a URL parameter?

Office Online passes the WOPI access token both as a URL parameter (called access_token) and in the Authorization header. This applies to all WOPI requests that originate from Office Online.

This is done primarily for compatibility reasons. Some host rely on the Authorization header because they are using an OAuth stack for creating and managing WOPI access tokens. Because WOPI does not define a way for a host to indicate that they are using OAuth, Office Online passes the access token both ways for maximum compatibility.

Tip: As a best practice, WOPI hosts should use the access token value from the URL parameter. This is the preferred way to pass access tokens, and not all WOPI clients will pass it in the Authorization header.

1.23.2 What is the maximum length of a WOPI access token?

There’s no enforced length limit on a WOPI access token; however, the overall URL length limit for Office Online is 2000 characters, and the access token is included on some GET requests between the Office Online browser apps and the Office Online service. This means that at some point, the access token can become so long that requests between the browser and the service will fail, which manifests as failing sessions. PowerPoint Online is particularly susceptible to this problem.

1.23.3 Office Online sometimes sends the access token in the query string when making requests to the service. Is this a problem?

If you monitor the traffic between the Office Online browser applications and the Office Online service, you may notice that some requests include the WOPI access token on the URL. We recognize that this is an issue and have designed our service Office Online logging infrastructure to deal with it. We scrub the query string from URLs before they are written to our logs. We have built a scrubber module into our standard server logging system that finds and redacts the access token in the two or three different forms it can take in a URL.

We believe that zero access tokens are currently being written to the Office Online service logs, and we consider it an urgent bug if we discover that they are.
1.23.4 How do I find the right URLs for the Office Online applications?

See *WOPI discovery*.

1.23.5 What authentication providers does Office Online support?

Office Online does not do any authentication, except as part of the *business user edit* flow. Hosts are expected to handle authentication and authorization by providing WOPI *access tokens*. All user-related information is provided to Office Online by the host using properties in *CheckFileInfo*.

1.23.6 What browsers does Office Online support?

See *Office Online browser support*.

1.23.7 How often should WOPI discovery be run?

See the *WOPI discovery* section for guidance regarding the frequency at which the WOPI discovery process should be run.

1.23.8 If I make an edit and immediately close the application, occasionally my edit is lost - why?

Office Online applications send edits from the browser to the server as often as possible. However, this process is not instantaneous and can be influenced by many factors including network latency and quality.

Office Online displays the save status in the bottom status bar:

![Figure 1.17: The ‘Saved to...’ UI for OneDrive](image)

If the status bar reads *Saved* or *Saved to <HOST NAME>*, then the edits have successfully made it to the server. However, if the status bar reads *Saving...* or *Working...*, then the edits have not yet been sent to the server and may be lost if the browser is closed or if you navigate away from the Office Online application immediately.

1.23.9 How does a WOPI host know when an editing session is finished?

While editing a file, a WOPI client will always maintain a WOPI lock on the file. When editing is complete, the file will be unlocked using the *Unlock* WOPI operation. Thus, once the file is successfully unlocked, the editing session is completed.

WOPI clients will always call *Unlock* at the end of an editing session, unless something happens that prevents the session from closing cleanly, such as a browser crash or a network dropout. In those cases the WOPI lock eventually times out, which is fundamentally equivalent to an explicit Unlock request.

See also:

In *co-authoring sessions*, the *Unlock* operation will only be called when the last user editing the document ends their session. Thus, if hosts wish to track what users are editing a document, they cannot rely completely on the locked state of the document. Hosts can use the *Edit_Notification* message to help gauge activity within a co-authoring session.
1.23.10 What are the file sizes supported by Office Online?

The explicit limits, where applicable, are listed in the table below. However, note that there is a 60-second file download time out that applies to all GetFile operations, and this time out can affect the perceived file size limit. In practice, this time out is rarely hit, since connectivity and bandwidth is typically very good between Office Online and host datacenters. However, hosts should be aware of this limit.

**Tip:** The FileUrl property can be set to change the URL that Office Online will use to download files from the host. This can be used to increase download speeds depending on the host’s architecture.

Code sample 1.14: File size limits

<table>
<thead>
<tr>
<th>Application</th>
<th>Mode</th>
<th>Limit</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Excel Online</td>
<td>View</td>
<td>5MB</td>
<td></td>
</tr>
<tr>
<td>Excel Online</td>
<td>Edit</td>
<td>5MB</td>
<td></td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>View</td>
<td>See notes</td>
<td>No limit, but subject to the 60 second time out for file downloads as described above.</td>
</tr>
<tr>
<td>PowerPoint Online</td>
<td>Edit</td>
<td>300MB</td>
<td>While the upper limit is 300MB, this is still subject to the overall 60 second time out for file downloads so it is possible that smaller files will hit that timeout.</td>
</tr>
<tr>
<td>Word Online</td>
<td>View</td>
<td>See notes</td>
<td>No limit, but subject to the 60 second time out for file downloads as described above.</td>
</tr>
<tr>
<td>Word Online</td>
<td>Edit</td>
<td>See notes</td>
<td>The technical limit is 100,000,000 (100 million) characters in the document XML; however, this does not correlate with file size in a meaningful way. For example, a 1000-page document, hundreds of MB in size does not hit this limit. For the vast majority of use-cases, this limit is irrelevant.</td>
</tr>
</tbody>
</table>

1.23.11 What are the IP ranges and ports used by Office Online?

Office Online does not provide IP ranges for partners to use to restrict traffic (i.e. IP-based ACLs). Office Online adds new servers and datacenters regularly and such IP lists will be out of date often. Hosts should use **proof keys** if they wish to verify that requests are coming from Office Online.

All WOPI communication is done using port 443, the standard HTTPS port.

1.23.12 What languages does Office Online support?

Office Online supports over 100 languages. In order to control the language of the Office Online UI, see the **UI_LLCC** placeholder value.

<table>
<thead>
<tr>
<th>Locale ID</th>
<th>Language</th>
<th>LCID</th>
</tr>
</thead>
<tbody>
<tr>
<td>af-ZA</td>
<td>Afrikaans</td>
<td>1078</td>
</tr>
<tr>
<td>am-ET</td>
<td>Amharic</td>
<td>1118</td>
</tr>
<tr>
<td>ar-SA</td>
<td>Arabic</td>
<td>1025</td>
</tr>
</tbody>
</table>

1.23. Frequently Asked Questions
Table 1.1 – continued from previous page

<table>
<thead>
<tr>
<th>Locale ID</th>
<th>Language</th>
<th>LCID</th>
</tr>
</thead>
<tbody>
<tr>
<td>as-IN</td>
<td>Assamese</td>
<td>1101</td>
</tr>
<tr>
<td>az-Latn-AZ</td>
<td>Azerbaijani (Latin)</td>
<td>1068</td>
</tr>
<tr>
<td>be-BY</td>
<td>Belarusian</td>
<td>1059</td>
</tr>
<tr>
<td>bg-BG</td>
<td>Bulgarian</td>
<td>1026</td>
</tr>
<tr>
<td>bn-BD</td>
<td>Bangla (Bangladesh)</td>
<td>2117</td>
</tr>
<tr>
<td>bn-IN</td>
<td>Bangla (India)</td>
<td>1093</td>
</tr>
<tr>
<td>bs-Latn-BA</td>
<td>Bosnian (Latin)</td>
<td>5146</td>
</tr>
<tr>
<td>ca-ES</td>
<td>Catalan</td>
<td>1027</td>
</tr>
<tr>
<td>ca-ES-valencia</td>
<td>Valencian</td>
<td>2051</td>
</tr>
<tr>
<td>chr-Cher-US</td>
<td>Cherokee</td>
<td>1116</td>
</tr>
<tr>
<td>cs-CZ</td>
<td>Czech</td>
<td>1029</td>
</tr>
<tr>
<td>cy-GB</td>
<td>Welsh</td>
<td>1106</td>
</tr>
<tr>
<td>da-DK</td>
<td>Danish</td>
<td>1030</td>
</tr>
<tr>
<td>de-DE</td>
<td>German</td>
<td>1031</td>
</tr>
<tr>
<td>el-GR</td>
<td>Greek</td>
<td>1032</td>
</tr>
<tr>
<td>en-US</td>
<td>English (US)</td>
<td>1033</td>
</tr>
<tr>
<td>es-ES</td>
<td>Spanish</td>
<td>3082</td>
</tr>
<tr>
<td>et-EE</td>
<td>Estonian</td>
<td>1061</td>
</tr>
<tr>
<td>eu-ES</td>
<td>Basque</td>
<td>1069</td>
</tr>
<tr>
<td>fa-IR</td>
<td>Persian (aka Farsi)</td>
<td>1065</td>
</tr>
<tr>
<td>fi-FI</td>
<td>Finnish</td>
<td>1035</td>
</tr>
<tr>
<td>fil-PH</td>
<td>Filipino</td>
<td>1124</td>
</tr>
<tr>
<td>fr-FR</td>
<td>French</td>
<td>1036</td>
</tr>
<tr>
<td>ga-IE</td>
<td>Gaelic Irish</td>
<td>2108</td>
</tr>
<tr>
<td>gd-GB</td>
<td>Scottish Gaelic</td>
<td>1084</td>
</tr>
<tr>
<td>gl-ES</td>
<td>Galician</td>
<td>1110</td>
</tr>
<tr>
<td>gu-IN</td>
<td>Gujarati</td>
<td>1095</td>
</tr>
<tr>
<td>ha-Latn-NG</td>
<td>Hausa (Latin)</td>
<td>1128</td>
</tr>
<tr>
<td>he-IL</td>
<td>Hebrew</td>
<td>1037</td>
</tr>
<tr>
<td>hi-IN</td>
<td>Hindi</td>
<td>1081</td>
</tr>
<tr>
<td>hr-HR</td>
<td>Croatian</td>
<td>1050</td>
</tr>
<tr>
<td>hu-HU</td>
<td>Hungarian</td>
<td>1038</td>
</tr>
<tr>
<td>hy-AM</td>
<td>Armenian</td>
<td>1067</td>
</tr>
<tr>
<td>id-ID</td>
<td>Indonesian</td>
<td>1057</td>
</tr>
<tr>
<td>ig-NG</td>
<td>Igbo</td>
<td>1036</td>
</tr>
<tr>
<td>is-IS</td>
<td>Icelandic</td>
<td>1039</td>
</tr>
<tr>
<td>it-IT</td>
<td>Italian</td>
<td>1040</td>
</tr>
<tr>
<td>ja-JP</td>
<td>Japanese</td>
<td>1041</td>
</tr>
<tr>
<td>ka-GE</td>
<td>Georgian</td>
<td>1079</td>
</tr>
<tr>
<td>kk-KZ</td>
<td>Kazakh</td>
<td>1087</td>
</tr>
<tr>
<td>km-KH</td>
<td>Khmer</td>
<td>1107</td>
</tr>
<tr>
<td>kn-IN</td>
<td>Kannada</td>
<td>1099</td>
</tr>
<tr>
<td>kok-IN</td>
<td>Konkani</td>
<td>1111</td>
</tr>
<tr>
<td>ko-KR</td>
<td>Korean</td>
<td>1042</td>
</tr>
<tr>
<td>ku-Arab-IQ</td>
<td>Central Kurdish</td>
<td>1170</td>
</tr>
<tr>
<td>ky-KG</td>
<td>Kyrgyz</td>
<td>1088</td>
</tr>
<tr>
<td>lb-LU</td>
<td>Luxembourgish</td>
<td>1134</td>
</tr>
<tr>
<td>lt-LT</td>
<td>Lithuanian</td>
<td>1063</td>
</tr>
<tr>
<td>lv-LV</td>
<td>Latvian</td>
<td>1062</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1.1 – continued from previous page

<table>
<thead>
<tr>
<th>Locale ID</th>
<th>Language</th>
<th>LCID</th>
</tr>
</thead>
<tbody>
<tr>
<td>mi-NZ</td>
<td>Maori</td>
<td>1153</td>
</tr>
<tr>
<td>mk-MK</td>
<td>Macedonian</td>
<td>1071</td>
</tr>
<tr>
<td>ml-IN</td>
<td>Malayalam</td>
<td>1100</td>
</tr>
<tr>
<td>mn-MN</td>
<td>Mongolian (Cyrillic)</td>
<td>1104</td>
</tr>
<tr>
<td>mr-IN</td>
<td>Marathi</td>
<td>1102</td>
</tr>
<tr>
<td>ms-MY</td>
<td>Malay</td>
<td>1086</td>
</tr>
<tr>
<td>mt-MT</td>
<td>Maltese</td>
<td>1082</td>
</tr>
<tr>
<td>nb-NO</td>
<td>Norwegian (Bokmal)</td>
<td>1044</td>
</tr>
<tr>
<td>ne-NP</td>
<td>Nepali</td>
<td>1121</td>
</tr>
<tr>
<td>nl-NL</td>
<td>Dutch</td>
<td>1043</td>
</tr>
<tr>
<td>nn-NO</td>
<td>Norwegian (Nynorsk)</td>
<td>2068</td>
</tr>
<tr>
<td>nso-ZA</td>
<td>Sesotho sa Leboa</td>
<td>1132</td>
</tr>
<tr>
<td>or-IN</td>
<td>Odia (aka Oriya)</td>
<td>1096</td>
</tr>
<tr>
<td>pa-Arab-PK</td>
<td>Punjabi (Arabic)</td>
<td>2118</td>
</tr>
<tr>
<td>pa-IN</td>
<td>Punjabi (Gurmukhi) (aka Punjabi (India))</td>
<td>1094</td>
</tr>
<tr>
<td>pl-PL</td>
<td>Polish</td>
<td>1045</td>
</tr>
<tr>
<td>prs-AF</td>
<td>Dari</td>
<td>1164</td>
</tr>
<tr>
<td>pt-BR</td>
<td>Portuguese (Brazil)</td>
<td>1046</td>
</tr>
<tr>
<td>pt-PT</td>
<td>Portuguese (Portugal)</td>
<td>2070</td>
</tr>
<tr>
<td>qut-GT</td>
<td>Kiche’</td>
<td>1158</td>
</tr>
<tr>
<td>quz-PE</td>
<td>Quechua</td>
<td>3179</td>
</tr>
<tr>
<td>ro-Ro</td>
<td>Romanian</td>
<td>1048</td>
</tr>
<tr>
<td>ru-Ru</td>
<td>Russian</td>
<td>1049</td>
</tr>
<tr>
<td>rw-RW</td>
<td>Kinyarwanda</td>
<td>1159</td>
</tr>
<tr>
<td>sd-Arab-PK</td>
<td>Sindhi</td>
<td>2137</td>
</tr>
<tr>
<td>si-LK</td>
<td>Sinhala</td>
<td>1115</td>
</tr>
<tr>
<td>sk-SK</td>
<td>Slovak</td>
<td>1051</td>
</tr>
<tr>
<td>sl-SI</td>
<td>Slovenian</td>
<td>1060</td>
</tr>
<tr>
<td>sq-AL</td>
<td>Albanian</td>
<td>1052</td>
</tr>
<tr>
<td>sr-Cyrl-BA</td>
<td>Serbian Cyrillic BiH</td>
<td>7194</td>
</tr>
<tr>
<td>sr-Cyrl-RS</td>
<td>Serbian (Cyrillic)</td>
<td>10266</td>
</tr>
<tr>
<td>sr-Latn-RS</td>
<td>Serbian (Latin)</td>
<td>9242</td>
</tr>
<tr>
<td>sv-SE</td>
<td>Swedish</td>
<td>1053</td>
</tr>
<tr>
<td>sw-KE</td>
<td>Kiswahili</td>
<td>1089</td>
</tr>
<tr>
<td>ta-IN</td>
<td>Tamil</td>
<td>1097</td>
</tr>
<tr>
<td>te-IN</td>
<td>Telugu</td>
<td>1098</td>
</tr>
<tr>
<td>tg-Cyrl-TJ</td>
<td>Tajik</td>
<td>1064</td>
</tr>
<tr>
<td>th-TH</td>
<td>Thai</td>
<td>1054</td>
</tr>
<tr>
<td>ti-ET</td>
<td>Tigrigna</td>
<td>1139</td>
</tr>
<tr>
<td>tk-TM</td>
<td>Turkmen</td>
<td>1090</td>
</tr>
<tr>
<td>tn-ZA</td>
<td>Setswana</td>
<td>1074</td>
</tr>
<tr>
<td>tr-TR</td>
<td>Turkish</td>
<td>1055</td>
</tr>
<tr>
<td>tt-RU</td>
<td>Tatar</td>
<td>1092</td>
</tr>
<tr>
<td>ug-CN</td>
<td>Uyghur</td>
<td>1152</td>
</tr>
<tr>
<td>uk-UA</td>
<td>Ukrainian</td>
<td>1058</td>
</tr>
<tr>
<td>ur-PK</td>
<td>Urdu</td>
<td>1056</td>
</tr>
<tr>
<td>uz-Latn-UZ</td>
<td>Uzbek</td>
<td>1091</td>
</tr>
<tr>
<td>vi-VN</td>
<td>Vietnamese</td>
<td>1066</td>
</tr>
<tr>
<td>wo-SN</td>
<td>Wolof</td>
<td>1160</td>
</tr>
</tbody>
</table>

Continued on next page
1.23.13 Why do I get an error when I try to view PDF documents in Office Online?

PDF viewing in Office Online is not included as part of the Cloud Storage Partner Program. We are in the process of deprecating the feature over the coming months, and are not supporting this integration as part of the Cloud Storage Partner Program. We are also deprecating this support for Microsoft products such as OneDrive. OneDrive no longer uses Office Online for PDF viewing.

Much has changed since Office Online originally added PDF viewing support. In particular, most modern browsers have excellent built-in PDF support within the browser, and most mobile devices have excellent native PDF support as well. These industry advances have made the reasons support was originally added obsolete.

1.23.14 How should users give feedback to Microsoft regarding Office Online?

Bugs/issues or compliments should be reported using the in-app feedback form, which can be launched via File → Help → Give Feedback or the Help Improve Office button on the status bar. The feedback form also has a link to UserVoice if the user chooses I have a suggestion.

1.24 Known Issues

1.25 Glossary

Broadcast  A broadcast is a special Office Online scenario where navigation through a document is driven by one or more presenters. A set of attendees can follow along with the presenter remotely.

See also:

present, attend

Host Page, Host Frame, Outer Frame  The host page (also called the ‘host frame’ or ‘outer frame’) is the HTML page which will host an iframe that points to an Office Online application.

See also:

Building a host page

Session Context  The Session Context is an optional parameter that a host can include on a WOPI request. It is a string, and is passed to Office Online in the sc query string parameter. If included on a WOPI request, Office Online will return the value of sc as the value of the X-WOPI-SessionContext HTTP header when making the CheckFileInfo and CheckFolderInfo WOPI requests.
1.26 Legal

1.26.1 Export Restrictions

Microsoft Technologies are subject to U.S. export jurisdiction. Company must comply with all applicable international and national laws, including the U.S. Export Administration Regulations, the International Traffic in Arms Regulations, Office of Foreign Assets Control sanctions programs, and end-user, end use and destination restrictions by U.S. and other governments related to Microsoft Technologies. For additional information, see http://www.microsoft.com/exporting.

1.27 Building this documentation locally

If you want to build this documentation locally, use the following steps:

1. Clone the repository.
2. Install Python 2.7 and pip. You can use the guide at http://www.tylerbutler.com/2012/05/how-to-install-python-pip-and-virtualenv-on-windows-with-powershell/ if needed. You can choose to install virtualenv if you wish, but you don’t have to.
3. Once Python and pip are installed, open up a PowerShell or cmd.exe prompt and go to the docs folder in this repository.
4. Type pip install -r requirements.txt. You should see some output like this:

```markdown
Downloading/unpacking Sphinx>=1.3.1 (from -r requirements.txt (line 1))
Downloading/unpacking sphinx-rtd-theme>=0.1.8 (from -r requirements.txt (line 2))
...
...
Successfully installed Sphinx sphinx-rtd-theme sphinxcontrib-findanything sphinxcontrib-httpdomain Jinja2 alabaster babel six Pygments snowballstemmer docutils colorama markupsafe
Cleaning up...
```

5. Run the command pip install sphinxcontrib-domaintools (this is necessary due to a bug in the sphinxcontrib-domaintools installer).
6. Now that all the prerequisites are installed, you can build the documentation using the following command: make.bat html. The built documentation will be output to build/html.

1.27.1 Checking spelling

If you want to check the spelling of the documentation, use the check_spelling.bat command. This will output a list of potentially misspelled words, along with the file in which the word was found and suggested replacement words. The output of the spell check will also be in the build/spelling/output.txt file:

```markdown
contributing\build_docs.rst:47: (spelling) ["spelling", "spell ling"]
contributing\build_docs.rst:52: (misspelled) ["misspelled", "dispelled", "mi spelled", "spelled", "mispelled"]
contributing\build_docs.rst:58: (misspelled) ["misspelled", "dispelled", "mi spelled", "spelled", "mispelled"]
```

Tip: The spell checker is not aware of reStructuredText includes, which are used often in the documentation. This means that the line numbers reported by the spell checker will likely be incorrect. In addition, spelling errors within
any included fragment will be reported as coming from the pages in which they’re included. If an included fragment is used in multiple pages, each page in which it is included will report the error.

Adding words to the ignore list

You can add words to the docs_source/spelling_wordlist.txt file to globally ignore the word as mis-spelled. Add a single word per line in alphabetical order.

Alternatively, you can use the spelling directive to add a list of known words to a specific file:

```plaintext
.. spelling::

   wopi
   CheckFileInfo
```

1.28 Office Online Documentation Style Guide

Attention: Sorry, this documentation hasn’t been written yet. You can track the status of issue #1 through our public GitHub issue tracker.

1.28.1 Whitespace guidelines

- Use spaces everywhere for whitespace. Do not use tabs.
- Use a single space in between sentences.
- Tab length: 4 spaces

1.28.2 Heading styles

The Office Online documentation should use the following characters for header underlines:

1. ===== (equals sign)
2. ------ (dashes)
3. ~~~~ (tildes)
4. ^^^^^ (carets)
5. """
6. ***** (asterisks)

Overlines should never be used.

Example

```plaintext
Header level 1
=============

Header level 2
```
1.28.3 Note/admonition styles

Note/admonition sections will be styled appropriately so they stand out from the rest of the text in a section. The standard reStructuredText directives such as .. note:: can be used, as well as some custom directives using the .. admonition:: directive. See the below examples for more information.

Examples

Note: This is a note using the .. note:: directive.
.. note::
   This is a note using the `.. note::` directive.

Tip: This is a tip using the .. tip:: directive.
.. tip::
   This is a tip using the `.. tip::` directive.

Warning: This is a warning using the .. warning:: directive.
.. warning::
   This is a warning using the `.. warning::` directive.

Danger: This is a danger message using the .. danger:: directive.
.. danger::
   This is a danger message using the `.. danger::` directive.

OneNote Online Note
This is an OneNote Online note using the .. admonition:: directive.
.. admonition:: OneNote Online Note

   This is an Office Online note using the `.. admonition::` directive.

Excel Online Note

This is an Excel Online note using the .. admonition:: directive.

.. admonition:: Excel Online Note

   This is an Excel Online note using the `.. admonition::` directive.

Office Online Tip

This is an Office Online tip using the .. admonition:: directive.

.. admonition:: Office Online Tip

   This is an Office Online tip using the `.. admonition::` directive.

Pre-release Content

This is a pre-release content warning using the .. admonition:: directive.

.. admonition:: Pre-release Content

   This is a pre-release content warning using the `.. admonition::` directive.
Index

A
App_LoadingStatus (global variable or constant), 48
attend
  WOPI Action, 16
attendservice
  WOPI Action, 16

B
Blur_Focus (global variable or constant), 45
Broadcast, 74
BUSINESS_USER, 18
C
ClosePostMessage, 42
cobalt
  WOPI Action Requirement, 16
containers
  WOPI Action Requirement, 17
convert
  WOPI Action, 15
D
DC_LLCC, 18
DISABLE_ASYNC, 18
DISABLE_BROADCAST, 18
DISABLE_CHAT, 18
E
edit
  WOPI Action, 15
Edit_Notification (global variable or constant), 49
EditModePostMessage, 43
editnew
  WOPI Action, 15
EditNotificationPostMessage, 43
EMBEDDED, 18
embedview
  WOPI Action, 15
F
File_Rename (global variable or constant), 49
FileSharingPostMessage, 43
formedit
  WOPI Action, 16
formssubmit
  WOPI Action, 15
FULLSCREEN, 18
G
getinfo
  WOPI Action, 15
Grab_Focus (global variable or constant), 46
H
Host Frame, 74
Host Page, 74
Host_PerfTiming (global variable or constant), 46
Host_PostmessageReady (global variable or constant), 47
I
imagepreview
  WOPI Action, 15
interactivepreview
  WOPI Action, 15
L
legacywebservice
  WOPI Action, 16
locks
  WOPI Action Requirement, 16
M
message (global variable or constant), 45
mobileView
  WOPI Action, 15
O
otherWindow.postMessage() (otherWindow method), 44
Outer Frame, 74
P
PERFSTATS, 18
PostMessageOrigin, 43
preloaded
    WOPI Action, 16
preloadview
    WOPI Action, 16
present
    WOPI Action, 16
presentservice
    WOPI Action, 16

R
RECORDING, 18
rest
    WOPI Action, 16
RFC
    RFC 1766, 18, 19
rtc
    WOPI Action, 16

S
Session Context, 74
syndicate
    WOPI Action, 16

T
THEME_ID, 19

U
ui-language (string), 48
UI_Close (global variable or constant), 50
UI_Edit (global variable or constant), 50
UI_LLCC, 19
UI_Sharing (global variable or constant), 50
update
    WOPI Action Requirement, 16

V
VALIDATOR_TEST_CATEGORY, 19
view
    WOPI Action, 15

W
wdUserSession (string), 48
WOPI Action
    attend, 16
    attendservice, 16
    convert, 15
    edit, 15
    editnew, 15
    embedview, 15
    formedit, 16
    formsubmit, 15
    getinfo, 15
    imagepreview, 15
    interactivepreview, 15
    legacywebservice, 16
    mobileView, 15
    preloadedit, 16
    preloadview, 16
    present, 16
    presentservice, 16
    rest, 16
    rtc, 16
    syndicate, 16
    view, 15
    WOPI Action Requirement
    cobalt, 16
    containers, 17
    locks, 16
    update, 16